MediaCmd SDK

Version 3.0

Drastic Technologies
523 The Queensway, Suite 102
Toronto, ON, M8Y 1J7
CANADA
(416) 255 5636
(Fax) 255 8780
http://www.drastictech.com

Contents

Introduction

The Drastic MediaCmd SDK is the mechanism by which all the elements of Drastic's DDRs communicate with one and other. This includes:

- Controlling Titan/VVW DDR Servers locally or remotely
- Controlling QuickClip locally
- Controlling QuickClip remotely with the network server option
- Controlling 9 pin serial VTRs and Servers via Sony, Odetics or VDCP protocol
- Receiving commands from 9 pin serial controller via Sony, Odetics or VDCP protocol
- Receiving commands from Drastic GUIs, servers and controllers
- Building HTML/Ajax status and control pages

MediaCmd is the communication method used within Drastic's DDR products. Any operation you see in a Drastic interface is available to your application through MediaCmd.

Overview

MediaCmd is a simple structure that supports a small, well defined set of commands for communicating transport, status and setup information between components in Drastic's DDR software. There are a number of fields in the stucture, but the important fields are:

* ctCmd – the primary command of this packet (Play, Pause, Stop, Record, etc)
* lSpeed – the transport speed for any play commands (integer where 65520 = normal forward play)
* dwPosition – the frame position for any play, pause or record commands
* dwStart – the starting frame for any play or record commands (inclusive)
* dwEnd – the ending frame for any play or record commands (exclusive)
* arbID – clip name, file name or other string/binary data for the command
* cfFlags – denotes which fields above are valid and their meaning

With the standard initialization of the structure, you can quickly build commands in this structure by changing a few members and sending it. The primary motion commands are ctPlay, ctPause, ctStop, ctRecStop, ctRecord, ctEject and ctTransfer. To get the current state (position, speed, start and end, current clip), the command ctGetState will return a filled in MediaCmd. For setup and less common status (e.g. video input, audio rms level, genlock) there is ctGetValue and ctSetValue. This is documented in the Low Level Header Docs.

Hopefully, you will not have to deal with the MediaCmd structure directly. The SDK includes a series of simple commands that should provide 99% of what your application needs. These functions are simply wrappers that create and send MediaCmd structures. The source for all these functions is provided in the SDK under SRC\General\vvwIF.cpp in case you need to modify or create new commands. The commands have slightly different names depending on which interface you use, but have the same root name, such as: Play(), PlayFromTo(), Stop(), Pause(), Seek(), Record() and UpdateStatus(). Commands

are also included for getting clip lists (GetNextClip()) and EDL elements from ::VTR_TC time code spaces (EDLResetToStart(), EDLGetEdit()). A selection of the most common settings are also included (SetVideoInput(), SetAudioInput(), SetVideoGenlock(), GetAudioPeakRMS(), etc). This interface is documented in the MediaCmd Documentation (previously called "VVW Interface Specification").

Installation

To properly work with the MediaCmd SDK, you should have a copy of the QuickClip software installed on your development system. Even if you target application will only use a part of the QuickClip software, it should all be installed for the development phase. Before proceeding with the SDK you should familarize your self with QuickClip's operation and toolset. All the elements available within QuickClip are the same elements available to your application through the SDK.

Once you have QuickClip installed, you should install the MediaCmd SDK. This will install the headers, libraries and source needed to control QuickClip from your application.

Choosing An Access Method

The SDK access method you should use depends on what you you would like your application to do, what programming language you are using and how involved you would like to/need to get in the low level MediaCmd structures. No matter which method you choose, the MediaCmd structure packets are exactly the same. Here are the main access method, with their pros and cons:

ActiveX – Direct On Server
Type: Microsoft ActiveX/COM access method
Pros: Easy to program, 1:1 relationship with QuickClip/XO interface.
Cons: Uses same config as QuickClip/XO. Requires a local copy of QuickClip.
Setup: Register VVW.DLL using RegSvr32.exe in the QuickClip installation directory.
Issues: Difficult to use when communicating via TCP/IP within the same machine. Can be overcome by using the default pipe communication system, but this requires changes for remote network control.

ActiveX – Network Remove
Type: Microsoft ActiveX/COM access method
Pros: Easy to program. No permanent config. Setup by user calls each time.
Cons: No permanent config. Setup by user calls each time.
Setup: Register vvwNet2.DLL using RegSvr32.exe.
Issues: Each connection must be setup through the ActiveX by the SDK caller

Ajax/HTTP – Standard Network
Type: Standard HTTP calls with XML Ajax returns
Pros: Easy to program. Realtime debug (firefox). Works with most browsers.
Cons: Can be slow in some instances, JavaScript somewhat limited.
Setup: Connect to server with you web browser
Issues: No permanent config beyond standard server

Java Direct – Network
Type: Network MediaCmd access via port 1234
Pros: Easy to program, some interface source included
Cons: Requires NDA, Drastic approval
Setup: Add package to project and call
Issues: Each connection must be setup by caller

Direct Link – no longer actively supported, use ActiveX
Type: Direct link to VVW.DLL
Pros: No ActiveX layer, code compatible with Linux, Irix, Mac OS-X.
Cons: Uses default config from QuickClip/XO, application must be run in QuickClip directory. Requires a local copy of QuickClip.
Setup: Link to vvw.lib, include vvw.h. Copy application into the QuickClip directory before running
Issues: Needs access to VVW.dll and all its support DLLs/D1Xs. Still needs to be setup by LocalConfig.exe or QuickClip/XO

Network DLL – no longer actively supported, use ActiveX
Type: Direct line to vvwNet2.dll
Pros: Consistant interface between local/remote and various OSs. Does not require a local copy of QuickClip.
Cons: Requires vvwNet2.dll and support dlls
Setup: Link to vvwNet2.lib, include vvwNet2.h. Copy dll set from SDK\bin directory with your application
Issues: Use the vvwOpenLocal function to avoid QuickClip configuration issues. Requires a few dlls to be added to you application installations. Does not run the client software automatically, so you application may need to start it, depending on what your application is doing.

Network Direct
Type: Direct compile of network sources in your app or your dll.
Pros: No extra dlls. Easy to customize and modify. Lots of comands already written.
Cons: You app needs to handle setup and may need to run QuickClip.exe/VVWServer.exe/QCRun.exe.
Setup: Copy source files from vvwNet2 into you project, modify and compile
Issues: Does not run the client software automatically, so you application may need to start it, depending on what your application is doing.

Manual
Type: Use the structures and defines to write your own communication and control layer.
Pros: This is required if you are using an unsupported development platform like PHP.
Cons: Everything has to be built and tested from the ground up.
Setup: None.
Issues: Unless you absolutely have to, this method is not recommended.

SDK Structure

The location of the SDK directories will depend on the location you choose during the installation, but the directories within there will always be the same:

* \BIN – Copies of the minimum dll set from a QuickClip installation.
* \LIB – Libraries required to link the vvwNet2.dll, examples and your application
* \INC – Header files required to compile vvwNet2.dll, examples and your application
* \Src\vwwNet2 – The source to our vvwNet2.dll from QuickClip
* \Src\General – Useful source files that do not compile into examples directly. The most important would be vvwIF.cpp that is the code behind the SDK functions described below.
* \Sample – Broken down into sub directories based on access type
    - \ActiveX – Examples that use the ActiveX control
    - \Direct – Examples that link directly to DLLs
    - \Java – Java based examples
    - \HTTP – Ajax based examples (must use QuickClip HTTP server to run)

Main Documentation

*** PDF version of the MediaCmd Documentation
http://www.drastictech.com/manuals/MediaCmdSDK.pdf

*** Online version of the MediaCmd Documentation
http://www.drastictech.com/manuals/MediaCmdSDK.html

*** Online wiki
http://www.drasticpreview.org/wakka.php?wakka=DrasticSoftwareSdks&v=131g

Low Level Header Documentation

*** Windows CHM help file version of the MediaCmd headers
http://www.drastictech.com/manuals/MediaCmd.chm

*** Online version of the MediaCmd headers
http://www.drastictech.com/manuals/mediacmd/

HTTP XML AJAX Documentation

*** Wikki area for HTTP XML MediaCmd
http://www.drasticpreview.org/wakka.php?wakka=DrasticHttpCommands&v=7sv

Components

The VVW interfaces are designed to allow VWW customers, OEMs and Drastic component OEMS to create custom control solutions in the simplest manner possible. We have attempted to create a series of similar interfaces across as many development environments as possible.

Controllable Components:

VVW	- Contains all other components
vvwXXX	- Hardware driver components for VVW-1000 through 7000 series DDRs
vvvNET2	- Direct network sender to VVW or user component (TCP/IP)
vvwDSync	- House clock/SMPTE LTC, Local GPI Control

Controlling Components:

vvwCTL	- RS-422/Network Interpreter for VTR/DDR Emulation
vvwNET2	- Direct network interpreter from VVW or user component (TCP/IP)
vvwHTTPD	- XML and Web based network control

Interface Types:

ActiveX	- MediaCmdX	- Visual C++, Visual Basic, Borland C/Delphi, etc
Java (direct)	- MediaCmdInterface	- Visual J++, Sun Java, Symantec
Unix	- mediacmd.so	- GNU (Linux/BSD)
Unix	- mediacmd.c	- Shrouded Source
HTML	- mediacmd.xml	- XML I/O (w HTML components)
XML	- mediacmd.xml	- XML direct network connection

Do not use this interface type (being removed):

VVW	- VVW.lib/VVW.h	- Direct DLL Control

References

MediaCmd.h	- Internal media cmd structures on which this interface is based.
VVWIF.h	- VVW Direct Access (includes previous access method as well)
VVWX.tlb	- MediaCmdX type library
MediaCmd.xml	- XML command set mirroring MediaCmd.h
SimpleVB/2	- Simple control from Visual Basic
SimlpleMFC/2	- Simple control from Visual C++ with MFC
CntrlVVW_Java	- Simple control from Visual J++/WFC
CntrlVVW_Cmd	- Command line controller using the DLL methods
VTRControl	- Test app for VTR control
LocalConfig	- VVW Setup
X-extdev.prm	- VTR plug in for Adobe Premiere
QuickClipProDVI	- Software only DDR

Interface

The following is assumed:

Language/Platform	Name	Notes
ActiveX VB	Vbx	ActiveX Control
ActiveX C++	Pcx	Pointer to ActiveX Control
Java Instantiation	Mci	MediaCmd Interface – Network Only
DLL		No prefix
Unix		No prefix – Network Only
XML		Via HTTP or Direct

- All pointers for ActiveX C++, DLL and Unix must be valid and correctly sized
- Any BSTR returns are freed by the caller using our utility function FreeString()
- Any char * returns are freed by the caller using our utility function FreeString()
- For Unix the type VVWBOOL is defined as typedef long VVWBOOL
- For ActiveX/Dll, BOOL is the Windows.h BOOL definition
- szClipName has a maximum of 8 alpha numeric characters as per the Louth and Odetics specifications
- szFileName is DOS/Windows formatted in one of two forms
  - X:\Some Path\On The Drive\Media Files Name.ext
  - \\VVWSERVER\X\ Some Path\On The Drive\Media Files Name.ext
- Not all channels are equally capable. Check functions before assuming they will work as they do on other VVW systems or channels. Each VVW System and associated channel will be consistent. Please check you server documentation for its specific features.

Nomenclature:

szClipName. Sz8CharClipName, etc

Any character area identified by the words clip and name refers to a Louth/Odetics style clip name. These names may be up to 8 characters in length plus a NULL terminating character. This means they should be allocated in the following manner:

```
Char szClipName[9] = "";          // Include unused 1 char safety
Char pClipName = new char[9];
Char szpClipName = malloc(9);
```

SzFileName, sz260FileName, etc

Any character area identified by the words file and name refers to a win32 style drive/path/file/ext name. These areas may be up to 260 characters in length plus a terminating NULL character. This means they should be allocated in the following manner:

```
Char szClipName[261] = "";        // Include unused 1 char safety
Char pClipName = new char[261];
```

Char szpClipName = malloc(261);

Time code strings

Time code strings may be as small as one character and have a maximum length of 14 characters plus a terminating NULL. This means they should be allocated in the following manner:

```
Char szClipName[15] = "";              // Include unused 1 char safety
Char pClipName = new char[15];
Char szpClipName = malloc(15);
```

JAVA

To use the Java VVW interface begin by importing the VVW interface package. The Java VVW Interface is neatly packaged in the VVWInterface.zip file provided.

```
importdrastic.mCmdIF.*;
```

You must ensure that the VVWInterface.zip file is accessible by the JVM. If the VVWInterface.zip file is not in the same file structure as your current project you must tell the JVM where to look for it by setting your classpath to it's current location.

After you have successfully imported the VVW interface package begin by instantiating a new instance of the MediaCmdIF class.

```
MediaCmdIFmci = new MediaCmdIF();
```

The newly instantiated MediaCmdIF object will have access to all VVW interface methods. Some MediaCmdIF methods require the user to instantiate a new object of the method's parameter type.

```
MediaCmdIF.ClipInfoclipData= new MediaCmdIF.ClipInfo(clipName)
mci.GetClipInfo(clipData);
long lEnd= clipInfo.lEnd;
```

This allows the VVWInterface to return multiple values in the defined class structure.

Note: Applets using the VVW Interface must have an archive tag in the applet's HTML file specifying the location of the VVWInterface.zip file.

Channel Control

EnableChannels

ActiveX VB	vbx.EnableChannels (lInternal0_31 As Long, lInternal32_63 As Long, lExternal64_95 As Long, lExternal96_127 As Long, lNetwork128_159 As Long, lNetwork160_192 As Long) As Long
ActiveX C++	long pcx->EnableChannels (long lInternal0_31, long lInternal32_63, long lExternal64_95, long lExternal96_127, long lNetwork128_159, long lNetwork160_192)
Java	*Not Available*
Dll	long vvw EnableChannels (long lInternal0_31, long lInternal32_63, long lExternal64_95, long lExternal96_127, long lNetwork128_159, long lNetwork160_192)
Unix	*Not Available*
XML	*Not Available*

Enable or disable channels based on the bit array supplied. VVW can contain up to 256 channels per access point. Channels 193-255 are disabled by default. The remaining channels may be enabled (if the corresponding bit is set to 1) or disabled (if the corresponding bit is set to 0) with this call. The first 64 channels (0 through 63) are reserved for internal ddr channels. Then next 64 channels (64 through 127) are reserved for VTR or DDR devices controlled via serial, Odetics or Louth protocol. The remaining channels are for controlling other devices through the network. Please note that a network channel controls all the channels on the network server box, so disabling one network connection may disable more than one channel. Always call GetMaxChannels() after setting the bits to make sure all the channels you expect exist actually exist. This should be the first call made to the activex component.

GetMaxChannels

ActiveX VB	vbx.GetMaxChannels () As Long
ActiveX C++	long pcx->GetMaxChannels ()
Java	long mci.GetMaxChannels ()
Dll	long vvwGetMaxChannels (long lChannel)
Unix	long GetMaxChannels ()
XML	*Not implemented*

Returns the maximum number of channels available for control. Channels start at 0 and end at max channels – 1. This return is one greater than the largest value available for SetCurChannel(), GetCurChannel() and the lChannel parameter for the DLL interface. This value will change if channels are being opened and closed via OpenChannel/CloseChannel, and every channel between 0 and GetMaxChannels() may not be active.

SetCurChannel

ActiveX VB	vbx.GetMaxChannels (lChannel As Long) As Long
ActiveX C++	long pcx->GetMaxChannels (long lChannel)
Java	long mci.SetMaxChannels (long lChannel)
Dll	*Not implemented*
Unix	long SetCurChannel (lChannel As long)
XML	Use 'setchannel=

Sets the channel to which all subsequent commands will be sent. This command does not exist in the Dll interface as the channel is sent on a per command basis.

Returns 0 or an error code

GetCurChannel

ActiveX VB	vbx.GetCurChannels () As Long
ActiveX C++	long pcx->GetCurChannels ()
Java	long mci.GetCurChannels ()
Dll	*Not implemented*
Unix	long GetCurChannel ()
XML	*Not required*

Get channel currently under control. This value will be between 0 and GetMaxChannels – 1.

GetCurChannelName

ActiveX VB	vbx.GetCurChannelName () As String
ActiveX C++	BSTR pcx->GetCurChannelName ()
Java	String mci.GetCurChannelName ()
Dll	Long vvwGetChannelName(long lChannel, char * szChanName)
Unix	long GetCurChannelName (char * szChanName)
XML	*Not required*

Get the name of the current channel. For unix and dll access, pass a null to get the channel name size, then pass in a pointer that points to a memory size of at least that many bytes (ANSI characters only).

GetCurChannelType

ActiveX VB	vbx.GetCurChannelType () As Long
ActiveX C++	long pcx->GetCurChannelType ()
Java	long mci.GetCurChannelType ()
Dll	long vvwGetChannelType (long lChannel)
Unix	long GetCurChannelType ()
XML	*Not supported*

Returns the basic type of the channel (VTR, Internal, User, House)

VVW_CHANTYPE_HOUSE	0x1	1
VVW_CHANTYPE_INTERNAL	0x2	2
VVW_CHANTYPE_VTR_DDR	0x4	4
VVW_CHANTYPE_UNKNOWN	0xFFFFFFFF	-1

ShowConfigDialog

ActiveX VB	vbx.ShowConfigDialog (hWnd as Long) As Long
ActiveX C++	long pcx->ShowConfigDialog (long hWnd)
Java	*Not Available*
Dll	long vvw ShowConfigDialog (long hWnd)
Unix	*Not Available*
XML	*Not Available*

Show the configuration dialog box for the current channel. If the channel does not have a configuration dialog, this function will return an error. It is not available in Java or unix as the dialog only shows up on the local machine, and cannot be seem through the network.

Network Specific

OpenChannel (vvwNet2 ActiveX)

ActiveX VB	vbx.OpenChannel (BSTR szAddress, long lPort, long lRemoteChannel) As Long
ActiveX C++	long pcx->OpenChannel (BSTR szAddress, long lPort, long lRemoteChannel)
Java	*Not implemented*
Dll	vvwOpenLocal(char * szServerAddress, long lPort, long lRemoteChannel)
Unix	vvwOpenLocal(char * szServerAddress, long lPort, long lRemoteChannel)
XML	*Not required*

Open a local channel to a remote (or localhost) server specified in szServerName (either the IP address or network name) using a specific port (default 1234) for a specific remote channel (default 0).

Returns local channel identifier if the connect was successful, else it returns -1. Please do not make any assumptions about what the local channel identifier will be, especially if you are opening and closing multiple channels. Closed identifiers may later get used for newly opened channels.

CloseChannel (vvwNet2 ActiveX)

ActiveX VB	vbx.OpenChannel (long lLocalChannel)
ActiveX C++	pcx->OpenChannel (long lLocalChannel)
Java	*Not implemented*
Dll	vvwClose (long lLocalChannel)
Unix	vvwClose (long lLocalChannel)
XML	*Not required*

Disconnect from a previously connected server using the local channel value returned by OpenChannel.

Connect

ActiveX VB	*Not implemented*
ActiveX C++	*Not implemented*
Java	boolean mci.Connect(String szServerAddress, int lPort)
Dll	*Not implemented*
Unix	Long Connect (char * szServerAddress, long lPort)
XML	*Not required*

Connect to a server specified in szServerName (either the IP address or network name) using a specific port (default 1234). For ActiveX and Dll access, please setup a persistent network connection in the VVW configuation utility.

Returns true if the connect was successful, else it returns false.

Disconnect

ActiveX VB	*Not implemented*
ActiveX C++	*Not implemented*
Java	boolean mci.Disconnect()
Dll	*Not implemented*
Unix	Long Disconnect()
XML	*Not required*

Disconnect from a previously connected server. For ActiveX and Dll access, please setup a persistent network connection in the VVW configuation utility.

Transport Operations

Play

ActiveX VB	vbx.Play () As Long
ActiveX C++	long pcx->Play ()
Java	long mci.Play ()
Dll	long vvwPlay (long lChannel)
Unix	long Play ()
XML	http://localhost/VVWXMLMediaCmd?Play

Play at normal speed.

Returns 0 if successful, else an error code.

PlayAtSpeed

ActiveX VB	vbx.PlayAtSpeed (lVVWSpeed As Long) As Long
ActiveX C++	long pcx->PlayAtSpeed (long lVVWSpeed)
Java	long mci.PlayAtSpeed (long lVVWSpeed)
Dll	long vvwPlayAtSpeed (long lChannel, long lVVWSpeed)
Unix	long PlayAtSpeed(long lVVWSpeed)
XML	http://localhost/VVWXMLMediaCmd?Play &speed=lVVWSpeed

Play at a particular VVW speed. VVW speeds use a base play speed of 65520. This means that play = 65520, reverse play = -65520, four times play = 262080, half play speed = 32760. Percentage play speeds may be converted to VVW speeds using the PercentageToVVWSpeed() function. For Speed calculations please see GetSpeed() below.

Returns 0 if successful, else an error code.

PlayFromTo

ActiveX VB	vbx.PlayFromTo (lFrom As Long, lTo As Long, fDeferred As Boolean) As Long
ActiveX C++	long pcx->PlayFromTo (long lFrom, long lTo, BOOL fDeferred)
Java	long mci.PlayFromTo (long lFrom, long lTo, boolean fDeferred)
Dll	long vvwPlayFromTo (long lChannel, long lFrom, BOOL lTo, bool fDeferred)
Unix	long PlayFromTo (long lFrom, long lTo, VVWBOOL fDeferred)
XML	http://localhost/VVWXMLMediaCmd?Play&start=lFrom &end=lTo

Play from a frame to another frame. As with editing systems, the 'from' point is included and will be displayed but the to point is NOT included and will not be displayed. This means that the last frame displayed will be lFrom – 1. The deferred flag allows

PlayFromTos to be stacked so that they will play back to back. The deferred flag in the status return should be false before another deferred command is added.

Returns 0 if successful, else an error code.

LoadClip

ActiveX VB	vbx.LoadClip (szClipName As String, lStartFrame As Long) As Long
ActiveX C++	long pcx-> LoadClip (BSTR szClipName, long lStartFrame)
Java	long mci. LoadClip (String szClipName, long lStartFrame)
Dll	long vvwLoadClip (long lChannel, char * szClipName, long lStartFrame)
Unix	long LoadClip (char * szClipName, long lStartFrame)
XML	http://localhost/VVWXMLMediaCmd?Pause &ClipID=szClipname

Clip Mode Only. Load a clip into the channel and display the lStartFrame.

Returns 0 if successful, else an error code.

PlayClip

ActiveX VB	vbx.PlayClip (szClipName As String, fDeferred As Boolean) As Long
ActiveX C++	long pcx->PlayClip (BSTR szClipName, BOOL fDeferred)
Java	long mci.PlayClip (String szClipName, boolean fDeferred)
Dll	long vvwPlayClip (long lChannel, char * szClipName, BOOL fDeferred)
Unix	long PlayClip(char * szClipName, VVWBOOL fDeferred)
XML	http://localhost/VVWXMLMediaCmd?Play &ClipID=szClipname&Flags=Deferred

Clip Mode Only. Play the entire clip specified by clip name. If the deferred flag is true, clip playback will only occur once the currently playing clip has finished. If there is no currently playing clip, playback will occur immediately.

Returns 0 if successful, else an error code.

PlayClipFromTo

ActiveX VB	vbx.PlayClipFromTo (szClipName As String, lFrom AS Long, lTo As Long, fDeferred As Boolean) As Long
ActiveX C++	long pcx->PlayClipFromTo (BSTR szClipName, long lFrom, long lTo, BOOL fDeferred)
Java	long mci.PlayClipFromTo (String szClipName, long lFrom, long lTo, boolean fDeferred)
Dll	long vvwPlayClipFromTo (long lChannel, char * szClipName, long lFrom, long lTo, BOOL fDeferred)
Unix	long PlayClipFromTo (char * szClipName, long lFrom, long lTo, VVWBOOL fDeferred)
XML	http://localhost/VVWXMLMediaCmd?Play&start=lFrom &end=lTo&ClipID=szClipname

Clip Mode Only. Play the specified portion of the clip specified by clip name. If the deferred flag is true, clip playback will only occur once the currently playing clip has finished. If there is no clip currently playing, playback will occur immediately.

Returns 0 if successful, else an error code.

PlayAtMs

ActiveX VB	vbx.PlayAtMs (lMsStart As Long) As Long
ActiveX C++	long pcx->PlayAtMs (long lMsStart)
Java	long mci.PlayAtMs (long lMsStart)
Dll	long vvwPlayAtMs (long lMsStart)
Unix	long PlayAtMs (long lMsStart)
XML	*Not supported*

Play starting at a particular millisecond time. Use the CurState to get the last frame aligned millisecond time from the controlled channel. Add a the millisecond equivalent of a number of frames and playback will commence at that time. Be sure and leave enough time for the command to be received and processed.

FastFoward

ActiveX VB	vbx.FastForward () As Long
ActiveX C++	long pcx-> FastForward ()
Java	long mci. FastForward ()
Dll	long vvwFastForward (long lChannel)
Unix	long FastForward ()
XML	http://localhost/VVWXMLMediaCmd?Play &speed=655200

Set the channel into its fastest possible forward motion state.
Returns 0 if successful, else an error code.

FastRewind

ActiveX VB	vbx.FastRewind () As Long
ActiveX C++	long pcx-> FastRewind ()
Java	long mci. FastRewind ()
Dll	long vvwFastRewind (long lChannel)
Unix	long FastRewind ()
XML	http://localhost/VVWXMLMediaCmd?Play &speed=-655200

Set the channel into its fastest possible reverse motion state.
Returns 0 if successful, else an error code.

Pause

ActiveX VB	vbx.Pause () As Long
ActiveX C++	long pcx-> Pause ()
Java	long mci. Pause ()
Dll	long vvwPause (long lChannel)
Unix	long Pause ()
XML	http://localhost/VVWXMLMediaCmd?Pause

Stop playback and display the current frame.

Returns 0 if successful, else an error code.

Seek

ActiveX VB	vbx.Seek (lFrame As Long) As Long
ActiveX C++	long pcx-> Seek (long lFrame)
Java	long mci. Seek (long lFrame)
Dll	long vvwSeek (long lChannel, long lFrame)
Unix	long Seek (long lFrame)
XML	http://localhost/VVWXMLMediaCmd?Pause &position=lFrame

Seek to a particular frame and display it to the user. This call will return before the seek is complete. Once the Position return in the status reaches the lFrame, the seek is complete.

Returns 0 if successful, else an error code.

SeekRelative

ActiveX VB	vbx.SeekRelative (lFrameOffset As Long) As Long
ActiveX C++	long pcx-> SeekRelative (long lFrameOffset)
Java	long mci. SeekRelative (long lFrameOffset)
Dll	long vvwSeekRelative (long lChannel, long lFrameOffset)
Unix	long SeekRelative (long lFrameOffset)
XML	http://localhost/VVWXMLMediaCmd?Pause &position=lFrameOffset

Seek a certain number of frames from the current position. Positive offsets imply forward direction, negative offset imply reverse.

Stop

ActiveX VB	vbx.Stop () As Long
ActiveX C++	long pcx-> Stop ()
Java	long mci. Stop ()
Dll	long vvwStop (long lChannel)
Unix	long Stop ()
XML	http://localhost/VVWXMLMediaCmd?Stop

Stop the output of the controlled channel and display the input video (not supported on all devices). On unsupported devices stop will be the same as a pause.

Returns 0 if successful, else an error code.

Record

ActiveX VB	vbx.Record () As Long
ActiveX C++	long pcx-> Record ()
Java	long mci. Record ()
Dll	long vvwRecord (long lChannel)
Unix	long Record ()
XML	http://localhost/VVWXMLMediaCmd?Record

Start the channel recording. In clip mode a default clip name will be used with a duration set to infinity. The record will stop on any transport command or at the point that the disk is full.

Returns 0 if successful, else an error code.

RecordAtMs

ActiveX VB	vbx.RecordAtMs (lMsStart As Long) As Long
ActiveX C++	long pcx->RecordAtMs (long lMsStart)
Java	long mci.RecordAtMs (long lMsStart)
Dll	long vvwRecordAtMs (long lMsStart)
Unix	long RecordAtMs (long lMsStart)
XML	*Not supported*

Record starting at a particular millisecond time. Use the CurState to get the last frame aligned millisecond time from the controlled channel. Add a the millisecond equivalent of a number of frames and the record will commence at that time. Be sure and leave enough time for the command to be received and processed.

RecordFromTo

ActiveX VB	vbx.RecordFromTo (lFrom As Long, lTo As Long) As Long
ActiveX C++	long pcx-> RecordFromTo (long lFrom, long lTo)
Java	long mci. RecordFromTo (long lFrom, long lTo)
Dll	long vwwRecordFromTo (long lChannel, long lFrom, long lTo)
Unix	long RecordFromTo (long lFrom, long lTo)
XML	http://localhost/VVWXMLMediaCmd?Record &start=lStart&end=lEnd

Record from a frame value to a frame value. As with editing systems, the 'from' point is included and will be recorded but the to point is NOT included and will not be recorded. This means that the last frame recorded will be lFrom – 1.

Returns 0 if successful, else an error code.

RecordStop (prepare record)

ActiveX VB	Vbx.RecordStop (szClipName As String, lDuration As Long) As Long
ActiveX C++	Long pcx-> RecordStop (BSTR szClipName, long lDuration)
Java	Long mci. RecordStop (String szClipName, long lDuration)
Dll	Long vvwRecordStop (long lChannel, char * szClipName, long lDuration)
Unix	Long RecordStop (char * szClipName, long lDuration)
XML	http://localhost/VVWXMLMediaCmd?RecStop

Clip Mode Only. Set the clip name and length of time to record in frames. The record will not actually start until Record() is called. If the lDuration is set to –1 the record will continue until Stop() is called or the channel runs out of space.

Returns 0 if successful, else an error code.

SetRecordPresets

ActiveX VB	vbx.SetRecordPresets (lVidEdit As Long, lAudEdit As Long lInfEdit As Long) As Long
ActiveX C++	long pcx-> SetRecordPresets (long lVidEdit, long lAudEdit, long lInfEdit)
Java	long mci. SetRecordPresets (long lVidEdit, long lAudEdit, long lInfEdit)
Dll	long vvwSetRecordPresets (long lVidEdit, long lAudEdit, long lInfEdit)
Unix	long SetRecordPresets (long lVidEdit, long lAudEdit, long lInfEdit)
XML	http://localhost/VVWXMLMediaCmd?Record &videochannels=lVidEdit&audiochannels=lAudEdit &infochannels=lInfEdit

Set the channels to record. Using –1 values implies that the Preset should be set to all available channels. Record Presets will remain set until the user changes them.

Returns 0 if successful, else an error code.

Eject

ActiveX VB	vbx.Eject () As Long
ActiveX C++	long pcx-> Eject ()
Java	long mci. Eject ()
Dll	long vvwEject (long lChannel)
Unix	long Eject ()
XML	http://localhost/VVWXMLMediaCmd?Eject

Eject the current media if it is removable. Normally only used with VTRs.

Returns 0 if successful, else an error code.

Special Commands

Please note: Not all the following commands are supported on all channels. Special restrictions may apply

Transfer

ActiveX VB	vbx.Transfer (lTargetChannel As Long, lPosition As Long, lStart As Long, lEnd As Long, lVidEdit As Long, lAudEdit As Long, lInfEdit As Long, szClipName As String, fToTape As Boolean) As Long
ActiveX C++	long pcx-> Transfer (long lTargetChannel, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOL fToTape)
Java	long mci. Transfer (long lTargetChannel, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, Boolean fToTape)
Dll	long vvwTransfer (long lChannel, long lTargetChannel, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOL fToTape)
Unix	long Transfer (long lTargetChannel, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, VVWBOOL fToTape)
XML	http://localhost/VVWXMLMediaCmd?Transfer &channel=lTargetChannel&position=lPosition &start=lStart&end=lEnd&videochannels=lVidEdit &audiochannels=lAudEdit&infochannels=lInfEdit &Flags=Invert

Transfer media from one channel to another. Only supported by VTR channels. Currently only implemented for VTR to internal channels or internal channels to VTR channels. To record media from a VTR, the fToTape should be false, to record media onto a VTR the fToTape should be true. The start and end point are from the playback device. The edit will occur at the current timecode location on the recorder.

Returns 0 if successful, else an error code.

Status Operation

UpdateStatus

ActiveX VB	vbx.UpdateStatus () As Long
ActiveX C++	long pcx-> UpdateStatus ()
Java	long mci. UpdateStatus ()
Dll	long vvwUpdateStatus (long lChannel)
Unix	long UpdateStatus ()
XML	http://localhost/VVWXMLGetStatus?

Retrieve the current status from the controlled device. The status is automatically updated by the interface, but this call ensures that the status is current when you are checking it.

Returns 0 if successful, else an error code.

VVWXMLGetStatus returns XML with a MediaCmd root element, for example:

```
  <?xml version="1.0" ?>
- <MediaCmd>
- <!--  Drastic MEDIACMD xml structure version 1,0
  -->
  <CmdID Value="-98238205" />
  <StructSize Value="336" />
  <Channel Value="-1" />
  <Cmd Value="1" UseClipID="1">Pause</Cmd>
  <Speed Value="0">0</Speed>
  <CmdAlt Value="2083947" TimeMs="1" />
  <Position Value="102" TcType="non-drop-frame"
UsingFrameCount="1">00:00:03:12</Position>
  <Start Value="0" TcType="non-drop-frame"
UsingFrameCount="1">00:00:00:00</Start>
  <End Value="2592000" TcType="non-drop-frame"
UsingFrameCount="1">24:00:00:00</End>
  <ClipID>::VTR_TC</ClipID>
  </MediaCmd>
```

GetState

ActiveX VB	vbx.GetState () As Long
ActiveX C++	long pcx-> GetState ()
Java	long mci. GetState ()
Dll	long vvwGetState (long lChannel)
Unix	long GetState ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current state

- ctStop 0 // Stop all action
- ctPause 1 // Pause, Seek
- ctPlay 2 // Play at specified speed (includes pause)
- ctRecord 3 // Record at specified speed
- ctRecStop 4 // Stop ready for recording
- ctEject 5 // Eject the current media
- ctError 17 // An error has occurred
- ctAbort 19 // Abort any queued commands

XML: See <MediaCmd> root element, <Cmd> sub-element (value)

GetFlags

ActiveX VB	vbx. GetFlags () As Long
ActiveX C++	long pcx-> GetFlags ()
Java	long mci. GetFlags ()
Dll	long vvwGetFlags (long lChannel)
Unix	long GetFlags ()
XML	Not required

Returns the current flags

- cfDeferred = 1, // 0x00000001 This is a delayed
- cfOverideDeferred = 1 << 30, // 0x40000000 Overide all previous deferred commands
- cfTimeMs = 1 << 1, // 0x00000002 Use Millisecond time for delayed time, not fields
- cfTimeTarget = 1 << 2, // 0x00000004 Delayed time is offset from current time code
- cfTimeHouseClock = 1 << 3, // 0x00000008 Delayed time is based on absolute (real) time
- cfUseSpeed = 1 << 4, // 0x00000010 Set the new speed
- cfUsePresets = 1 << 5, // 0x00000020 Use video and audio edit presets
- cfUsePosition = 1 << 6, // 0x00000040 Use the position setting
- cfUsePositionOffset = 1 << 7, // 0x00000080 Position is an offset
- cfUseStart = 1 << 8, // 0x00000100 Start a new timecode
- cfUseStartOffset = 1 << 9, // 0x00000200 Start is an offset from current tc
- cfUseEnd = 1 << 10, // 0x00000400 End command as specified
- cfUseEndOffset = 1 << 11, // 0x00000800 End is and offset from current tc
- cfUseAllIDs = 1 << 12, // 0x00001000 Use all clip IDs

- cfUseClipID = 1 << 13, // 0x00002000 Use new clip ID, otherwise use last or none
- cfNoClipFiles = 1 << 14, // 0x00004000 Use new clip ID, otherwise use last or none
- cfNoTCSpaces = 1 << 15, // 0x00008000 Use new clip ID, otherwise use last or none
- cfUseCmdAlt = 1 << 16, // 0x00010000 Use the dwCmdAlt
- cfIsShuttle = 1 << 17, // 0x00020000 Use speed in play for shuttle
- cfFields = 1 << 20, // 0x00100000 Position, start and end are fields, not frames
- cfRipple = 1 << 21, // 0x00200000 Ripple for insert or delete
- cfLoop = 1 << 22, // 0x00400000 Loop the clip or in out
- cfTrigger = 1 << 23, // 0x00800000 Trigger using dsync class
- cfPreview = 1 << 24, // 0x01000000 Preview set (EE, non rt play)
- cfInvert = 1 << 28, // 0x10000000 Invert a transfer
- cfTest = 1 << 29, // 0x20000000 See if the command exists
- cfNoReturn = 1 << 31, // 0x80000000 No return mediacmd is required

GetSpeed

ActiveX VB	vbx. GetSpeed () As Long
ActiveX C++	long pcx-> GetSpeed ()
Java	long mci. GetSpeed ()
Dll	long vvwGetSpeed (long lChannel)
Unix	long GetSpeed ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current VVW speed if the cfUseSpeed flag is set, otherwise pause or full play speed. VVW speeds are based on 65520 as the play speed. To translate to decimal number where 1.0 represents play, use the following formula:

D1Speed = ((double)VVWSpeed / 65520.0)

For percentages, where 100.0 represents play speed, use the following formula:

Dpercent = (((double)VVWSpeed * 100.0) / 65520.0)
= ((double)VVWSpeed / 655.2)

XML: See <MediaCmd> root element, <Speed> sub-element

Typical VVW speeds (note speeds are linear):

Pause	0%	0
Play	100%	65520
Half Play	50%	32760
Reverse Play	-100%	-65520
Reverse Double Play	-200%	131040
10 Time Forward Play	1000%	655200
Max Forward Play	90000%	5896800
Max Reverse Play	-90000%	-5896800

GetPosition

ActiveX VB	vbx. GetPosition () As Long
ActiveX C++	long pcx-> GetPosition ()
Java	long mci. GetPosition ()
Dll	long vvwGetPosition (long lChannel)
Unix	long GetPosition ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current position if the cfUsePosition flag is set, otherwise invalid.
XML: See <MediaCmd> root element, <Position> sub-element (value)

GetLastMs

ActiveX VB	vbx. GetLastMs () As Long
ActiveX C++	long pcx-> GetLastMs ()
Java	long mci. GetLastMs ()
Dll	long vvwGetLastMs (long lChannel)
Unix	long GetLastMs ()
XML	http://localhost/VVWXMLGetStatus?

Returns the millisecond time the last status occurred (time of the last vertical blank).
XML: See <MediaCmd> root element, <CmdAlt> sub-element

GetStart

ActiveX VB	vbx. GetStart () As Long
ActiveX C++	long pcx-> GetStart ()
Java	long mci. GetStart ()
Dll	long vvwGetStart (long lChannel)
Unix	long GetStart ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current start or in point if the cfUseStart flag is set.
XML: See <MediaCmd> root element, <Start> sub-element

GetEnd

ActiveX VB	vbx. GetEnd () As Long
ActiveX C++	long pcx-> GetEnd ()
Java	long mci. GetEnd ()
Dll	long vvwGetEnd (long lChannel)
Unix	long GetEnd ()
XML	http://localhost/VVWXMLGetStatus?

Return the current end point or out point if cfUseEnd is set.
XML: See <MediaCmd> root element, <End> sub-element

GetClipName

ActiveX VB	vbx. GetClipName () As String
ActiveX C++	BSTR pcx-> GetClipName ()
Java	String mci. GetClipName ()
Dll	long vvwGetClipName (long lChannel, char * sz8CharClipName)
Unix	Char * GetClipName ()
XML	http://localhost/VVWXMLGetStatus?

Only supported in clip Mode. Returns the current clip name, if any. For dll access, the memory must be at least 9 bytes long (8 character bytes + NULL) and is always ANSI.
XML: See <MediaCmd> root element, <CmdID> sub-element

GetFileName

ActiveX VB	Vbx. GetFileName () As String
ActiveX C++	BSTR pcx-> GetFileName ()
Java	String mci. GetFileName ()
Dll	Long GetFileName (long lChannel, char * sz260CharFileName)
Unix	Char * GetFileName ()
XML	Not supported

Returns the current file name, if any. For dll access, the memory must be at lease 261 bytes long (260 bytes max path + NULL) and is always ANSI.

GetCurTC

ActiveX VB	Vbx. GetCurTC () As String
ActiveX C++	BSTR pcx-> GetCurTC ()
Java	String mci. GetCurTC ()
Dll	Long GetCurTC (long lChannel, char * sz14ByteTC)
Unix	Char * GetCurTC ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current time code as a string (e.g. "00:01:00:00"). For dll access, the memory must always be at least 15 bytes long (14 byte time code plus id + NULL) and is always ANSI.
XML: See <MediaCmd> root element, <Positon> sub-element (text)

GetCurState

ActiveX VB	Vbx. GetCurState () As String
ActiveX C++	BSTR pcx-> GetCurState ()
Java	String mci. GetCurState ()
Dll	Long GetCurState (long lChannel, char * sz14ByteState)
Unix	Char * GetCurState ()
XML	http://localhost/VVWXMLGetStatus?

Returns the current state as a string (e.g. "Play"). For dll access, the memory must always be at least 15 bytes long (14 byte state + NULL) and is always ANSI.
XML: See <MediaCmd> root element, <Cmd> sub-element (text)

Media Operations

Clip Mode Operations

GetNextClip

ActiveX VB	Vbx. GetNextClip (szLastClip As String) As String
ActiveX C++	BSTR pcx-> GetNextClip (BSTR szLastClip)
Java	String mci. GetNextClip (String szLastClip)
Dll	Char * vvwGetNextClip (long lChannel, char * sz8CharLastClipCurClip)
Unix	Char * GetNextClip (char * sz8CharLastClipCurClip)
XML	http://localhost/VVWXMLNextClip?

Clip Mode Only. Returns the next clip identifier. To get the first clip, szLastClip should be an empty string. Once the last clip available has been returned, GetNextClip will return an error or NULL for unix/dll access. Please note: For unix/dll access, the sz8CharLastClipCurClip memory area is used for the new clip. The previous clip name is therefore lost and the memory is not allocated by the vvw.

Returns 0 if successful, else an error code.

VVWXMLNextClip returns XML with a ClipInfo root element, for example:

```
<?xml version="1.0" ?>
- <ClipInfo>
- <!--  Drastic ClipInfo xml structure version 1,0
 -->
 <ClipID>::Test</ClipID>
 <FileName>::Test</FileName>
 <Start Value="0" TcType="non-drop-frame">00:00:00:00</Start>
 <End Value="0" TcType="non-drop-frame">02:00:00:00</End>
 </ClipInfo>
```

GetClipInfo

ActiveX VB	Vbx. GetClipInfo (szClipName As String, ByRef lStart As Long, ByRef lEnd As Long, ByRef lVidEdit As Long, ByRef lAudEdit As Long, ByRef lInfEdit As Long, szFileName As String) As Long
ActiveX C++	Long pcx-> GetClipInfo (BSTR szLastClip, long * lStart, long * lEnd, long * lVidEdit, long * lAudEdit, long * lInfEdit, BSTR * szFileName)
Java	Long mci. GetClipInfo (mci.ClipInfo clipData)
Dll	Long vvwGetClipInfo (long lChannel, char * sz8CharClipName, long * lStart, long * lEnd, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * sz260CharFileName)
Unix	Long GetClipInfo (char * sz8CharClipName, long * lStart, long * lEnd, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * sz260CharFileName)
XML	http://localhost/VVWXMLNextClip?

Retuns the basic information from szClip. The information is located in lStart, lEnd, lVidEdit, lAudEdit and szFileName as the in point, out point, number of video channels, number of audio channels, and the file name respectively.

Returns 0 if successful, else an error code.

Java: This method requires the user to instantiate a new object of type ClipInfo. The sz8CharClipName, lStart, lEnd, lVidEdit, lAudEdit, lInfEdit, and sz260CharFileName values are returned in the oject's instance variables.

XML: returns <ClipInfo> root element, <ClipID>, <FileName>, <Start>, <End> sub elements

GetClipInfoEx

ActiveX VB	vbx. GetClipInfoEx (szClipName As String, ByRef lCreation As Long, ByRef lLastModification As Long, ByReg lFileSize As Long, ByRef lDiskFragments) As Long
ActiveX C++	Long pcx-> GetClipInfoEx (BSTR szClipName, long * lCreation, long * lLastModification, long * lFileSize, long * lDiskFragments)
Java	*Not Implemented*
Dll	Long vvwGetNextClipEx (long lChannel, char * szClipName, long * lCreation, long * lLastModification, long * lFileSize, long * lDiskFragments)
Unix	Long GetNextClipEx (char * szClipName, long * lCreation, long * lLastModification, long * lFileSize, long * lDiskFragments)
XML	Not supported

Retuns the extended information from szClip. The information is located in lStart, lEnd, lVidEdit, lAudEdit and szFileName as time of creation, last modified date, the file size, and the number of fragments in the file respectively.

Returns 0 if successful, else an error code.

CopyClip

ActiveX VB	vbx. CopyClip (szSourceClip As String, szDestClip As String, lStart As Long, lEnd As Long) As long
ActiveX C++	Long pcx-> CopyClip (BSTR szSourceClip, BSTR szDestClip, long lStart, long lEnd)
Java	Long mci. CopyClip (String szSourceClip, String szDestClip, long lStart, long lEnd)
Dll	Long vvwCopyClip (long lChannel, char * szSourceClip, char * szDestClip, long lStart, long lEnd)
Unix	Long CopyClip (char * szSourceClip, char * szDestClip, long lStart, long lEnd)
XML	Not supported

Create a virtual copy of a clip, changing the in and out points if necessary. To use the whole clip, set lStart to 0 and the end to –1.

Returns 0 if successful, else an error code.

VTR Mode Operations

EDLResetToStart

ActiveX VB	Vbx. EDLResetToStart () As Long
ActiveX C++	Long pcx-> EDLResetToStart ()
Java	Long mci. EDLResetToStart ()
Dll	Long vvwEDLResetToStart (long lChannel)
Unix	Long EDLResetToStart ()
XML	Not supported

Reset the edl returns in VTR mode to the first element of the list.

EDLGetEdit

ActiveX VB	Vbx. EDLGetEdit (ByRef lRecordIn As Long, ByRef lPlayIn As Long, ByRef lPlayOut As Long, ByRef lVidEdit As Long, ByRef lAudEdit As Long, ByRef lInfEdit As Long, ByRef szClipName As String, ByRef szFileName As Long) As Long
ActiveX C++	Long pcx-> EDLGetEdit (long * lRecordIn, long * lPlayIn, long * lPlayOut, long * lVidEdit, long * lAudEdit, long * lInfEdit, BSTR * szClipName, BSTR * szFileName)
Java	long mci. EDLGetEdit (VTREditLine editInfo)
Dll	long vvwEDLGetEdit (long lChannel, long * lRecordIn, long * lPlayIn, long * lPlayOut, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * sz8CharClipName, char * sz260CharFileName)
Unix	Long EDLGetEdit (long * lRecordIn, long * lPlayIn, long * lPlayOut, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * szClipName, char * szFileName)
XML	http://localhost/VVWXMLEDLInfo?position=0 &videochannels=0&audiochannels=0&infochannels=0

Returns an edit line from the VTR space of an internal channel. The function will continue to return the next edit in the timecode space until the last edit is returned, after which an error will be returned. To reset to the start of the Edl use EDLResetToStart.

Returns 0 if successfule else an Error code.

Java: This method requires the user to instantiate a new object of type VTREditLine. The lRecordIn, lPlayIn, lPlayOut, lVidEdit, lAudEdit, lInfEdit, szClipName, and szFileName values are returned in the oject's instance variables of the same name.

VVWXMLEDLInfo returns XML with a <MediaCmd> root element, for example:

```
<?xml version="1.0" ?>
- <MediaCmd>
  - <!--  Drastic MEDIACMD xml structure version 1,0
  -->
  <CmdID Value="-98238205" />
  <StructSize Value="336" />
  <Channel Value="0" />
```

```
  <Cmd Value="14" UseClipID="1">GetValue</Cmd>
  <VideoChannels Value="1" />
  <AudioChannels Value="0" />
  <InfoChannels Value="0" />
  <CmdAlt Value="93" />
  <Position Value="5" TcType="non-drop-frame">00:00:00:05</Position>
  <Start Value="0" TcType="non-drop-frame">00:00:00:00</Start>
  <End Value="5" TcType="non-drop-frame">00:00:00:05</End>
  <FileName>V:\Drastic Base Media\avi_er001_720x486_YUY2.avi</FileName>
</MediaCmd>
```

Shared Operations

GetLastChangeMs

ActiveX VB	vbx. GetLastChangeMs () As Long
ActiveX C++	long pcx-> GetLastChangeMs ()
Java	long mci. GetLastChangeMs ()
Dll	long vvwGetLastChangeMs (long lChannel)
Unix	long GetLastChangeMs ()
XML	Not supported

Returns the millisecond time of the last change in the current mode (clip or vtr).

Insert

ActiveX VB	vbx. Insert (szClipName As String, szFileName As String, lPosition As Long, lStart As Long, lEnd As Long, lVidEdit As Long, lAudEdit As Long, lInfEdit As Long, fRipple As Boolean) As Long
ActiveX C++	long pcx-> Insert (BSTR szClipName, BSTR szFileName, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Java	long mci. Insert (String szClipName, String szFileName, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, Boolean fRipple)
Dll	long vvwInsert (long lChannel, char * szClipName, char * szFileName, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Unix	long Insert (char * szClipName, char * szFileName, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, VVWBOOl fRipple)
XML	http://localhost/VVWXMLMediaCmd?Insert &ClipID=szClipName&position=lPosition &start=lStart&end=lEnd&videochannels=lVidEdit &audiochannels=lAudEdit&infochannels=lInfEdit &Flags=Ripple

Internal Channels Only. Do not use yet.

Blank

ActiveX VB	vbx. Blank (szClipName As String, lStart As Long, lEnd As Long, lVidEdit As Long, lAudEdit As Long, lInfEdit As Long, fRipple As Boolean) As Long
ActiveX C++	long pcx-> Blank (BSTR szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Java	long mci. Blank (String szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, Boolean fRipple)
Dll	long vvwBlank (long lChannel, char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Unix	long Blank (char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, VVWBOOl fRipple)
XML	http://localhost/VVWXMLMediaCmd?Blank &ClipID=szClipName&position=lPosition &start=lStart&end=lEnd&videochannels=lVidEdit &audiochannels=lAudEdit&infochannels=lInfEdit &Flags=Ripple

Internal Channels Only. Do not use yet.

Delete

ActiveX VB	vbx. Delete (szClipName As String, lStart As Long, lEnd As Long, lVidEdit As Long, lAudEdit As Long, lInfEdit As Long, fRipple As Boolean) As Long
ActiveX C++	long pcx-> Delete (BSTR szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Java	long mci. Delete (String szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, Boolean fRipple)
Dll	long vvwDelete (long lChannel, char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Unix	long Delete (char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, VVWBOOl fRipple)
XML	http://localhost/VVWXMLMediaCmd?Delete &ClipID=szClipName&position=lPosition &start=lStart&end=lEnd&videochannels=lVidEdit &audiochannels=lAudEdit&infochannels=lInfEdit &Flags=Ripple

Internal Channels Only. Do not use yet.

Trim

ActiveX VB	vbx. Trim (lPosition As Long, lStartOffset As Long, lEndOffset As Long, lVidEdit As Long, lAudEdit As Long, lInfEdit As Long, fRipple As Boolean) As Long
ActiveX C++	long pcx-> Trim (long lPosition, long lStartOffset, long lEndOffset, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Java	long mci. Trim (long lPosition, long lStartOffset, long lEndOffset, long lVidEdit, long lAudEdit, long lInfEdit, Boolean fRipple)
Dll	long vvwTrim (long lChannel, long lPosition, long lStartOffset, long lEndOffset, long lVidEdit, long lAudEdit, long lInfEdit, BOOl fRipple)
Unix	long Trim (long lPosition, long lStartOffset, long lEndOffset, long lVidEdit, long lAudEdit, long lInfEdit, VVWBOOl fRipple)
XML	http://localhost/VVWXMLMediaCmd?Trim &position=lPosition&start=lStartOffset &end=lEndOffset&videochannels=lVidEdit &audiochannels=lAudEdit&infochannels=lInfEdit &Flags=Ripple

Internal Channels Only. Do not use yet.

Settings

The 'Value' commands allow settings to be changed on a particular channel. The most common settings have been made into functions, but all settings use ValueSupported, ValueMin, ValueMax, ValueGet and ValueSet. Most applications will only require the functions below. If an extended settings is required, please see the MediaCmd reference.

ValueSupported

ActiveX VB	vbx. ValueSupported (lValueType As Long) As Long
ActiveX C++	long pcx-> ValueSupported (long lValueType)
Java	long mci. ValueSupported (long lValueType)
Dll	long vvwValueSupported (long lChannel, long lValueType)
Unix	long ValueSupported (long lValueType)
XML	http://localhost/ValueSupported&cmdalt=valuetype &position=lValueType

Returns the supported attributes of a get/set value (gsClipMode, gsTcSource, etc) or –1 for not supported.

ValueGet

ActiveX VB	vbx. ValueGet (lValueType As Long, ByRef lMin As Long, ByRef lMax As Long) As Long
ActiveX C++	long pcx-> ValueGet (long lValueType, long * plMin, long * plMax)
Java	long mci. ValueGet (GetValueMcmd mCmdValues)
Dll	long vvwValueGet (long lChannel, long lValueType, long * plMin, long * plMax)
Unix	long ValueGet (long lValueType, long * plMin, long * plMax)
XML	http://localhost/GetValue&cmdalt=valuetype &position=lValueType

Returns the current setting for a get/set value.

Java: This method requires the user to instantiate a new object of type GetValueMcmd. The plMin and plMax values are returned in the oject's instance variables: plMin and plMax.

ValueSet

ActiveX VB	vbx. ValueSet (lValueType As Long, lSetting As Long) As Long
ActiveX C++	long pcx-> ValueSet (long lValueType, long lSetting)
Java	long mci. ValueSet (long lValueType, long lSetting)
Dll	long vvwValueSet (long lChannel, long lValueType, long lSetting)
Unix	long ValueSet (long lValueType, long lSetting)
XML	http://localhost/SetValue&cmdalt=valuetype &position=lSetting

Sets the get/set value to setting.

ValueSet2

ActiveX VB	vbx. ValueSet2 (lValueType As Long, lSetting As Long, lStart As Long, lEnd As Long, lVidChannel As Long, lAudChannel As Long, lInfChannel As Long) As Long
ActiveX C++	long pcx-> ValueSet2 (long lValueType, long lSetting, long lStart, long lEnd, long lVidChannel, long lAudChannel, long lInfChannel)
Java	long mci. ValueSet2 (long lValueType, long lSetting, long lStart, long lEnd, long lVidChannel, long lAudChannel, long lInfChannel)
Dll	long vvwValueSet2 (long lChannel, long lValueType, long lSetting, long lStart, long lEnd, long lVidChannel, long lAudChannel, long lInfChannel)
Unix	long ValueSet2 (long lValueType, long lSetting, long lStart, long lEnd, long lVidChannel, long lAudChannel, long lInfChannel)
XML	Not supported – see ValueSet

Sets the get/set value to setting with extended parameters. Please set unused parameters to NULL.

Settings Format

All the settings, except where noted, have the following format:

ActiveX VB	Vbx. Get*XXX* () As Long
	Vbx. Set*XXX* (lSetting As Long) As Long
ActiveX C++	Long pcx-> Get*XXX* ()
	Long pcx-> Set*XXX* (long lSetting)
Java	Long mci. Get*XXX* ()
	Long mci. Set*XXX* (long lSetting)
Dll	Long vvwGet*XXX* ()
	Long vvwSet*XXX* (long lSetting)
Unix	Long Get*XXX* ()
	Long Set*XXX* (long lSetting)

Base Settings

Get/SetClipMode

Calls ValueXXX with gsClipMode. If equal to 1 then the channel is in clip mode, if 0 the channel is in VTR mode.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=clipmode&position=0)

Get/SetTCType

Calls ValueXXX with gsTcType (drop frame, non drop frame, pal).

```
#define TC2_TCTYPE_MASK          0x000000FF
#define TC2_TCTYPE_FILM          0x00000001    // 24 fps
#define TC2_TCTYPE_NDF           0x00000002    // NTSC Non Drop Frame
#define TC2_TCTYPE_DF            0x00000004    // NTSC Drop Frame
#define TC2_TCTYPE_PAL           0x00000008    // PAL
#define TC2_TCTYPE_50            0x00000010    // PAL (double rate)
#define TC2_TCTYPE_5994          0x00000020    // NTSC 59.94fps 720p
#define TC2_TCTYPE_60            0x00000040    // NTSC 60fps 720p
#define TC2_TCTYPE_NTSCFILM      0x00000080    // NTSC FILM 23.97
```

(XML: localhost/XMLMediaCmd?SetValue&cmdalt=tctype&position=2)

Get/SetTCSource

Calls ValueXXX with gsTcSource (VITC, LTC, Control, Clip).

```
//! For cmdGetSetValue::gsTcSource - Using LTC
#define GS_TCSOURCE_LTC          1
//! For cmdGetSetValue::gsTcSource - Using VITC
#define GS_TCSOURCE_VITC         2
//! For cmdGetSetValue::gsTcSource - Using CTL
#define GS_TCSOURCE_CTL          4
//! For cmdGetSetValue::gsTcSource - Using absolute clip
#define GS_TCSOURCE_CLIP         7
```

(XML: localhost/XMLMediaCmd?GetValue&cmdalt=tcsource)

Get/SetAutoMode

Calls ValueXXX with gsAutoMode. Required for play lists, deferred commands and auto edit commands on VTRs.

GetAvailablePresets

ADD FUNCTIONS lVidEdit, lAudEdit, lInfEdit

Returns the supported audio, video and info presets for a channel.

Java: Values for audio, video and info presets are returned in variable members of the class mci.AvailablePresets.

Audio Settings

Get/SetAudioInput

ADD FUNCTION lAudIn

Get the current audio input.

Java: Must indicate the channel(s) to get/set. This is achieved by sending a long value representing the channel(s) you wish to get/set. For example if you wish to set audio channels 1 and 4 set the audChannels paramater to 9 (1001). You may also use the predefined audio channel defenitions in the MEDIACMD structure (audChanX, where x is the audio channel - 1).

//! Audio in/out unbalanced (RCA connector) high impedence at -10db (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_10 0x001
//! Audio in/out unbalanced (RCA connector) high impedence at -4db (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_4 0x002
//! Audio in/out balanced (XLR connector) 600ohm impedence at -10db (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_BALANCED_10 0x010
//! Audio in/out balanced (XLR connector) 600ohm impedence at +4db (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_BALANCED_4 0x020
//! Audio in/out digital single wire (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_SPDIF 0x100
//! Audio in/out digital balanced with clock (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_AES_EBU 0x200
//! Audio in/out embedded in SDI or HD-SDI video signal (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_EMBEDDED 0x400
//! No audio in/out available, or cannot be configured (cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_NONE 0
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsAudInSelect&position=lSetting &videochannels=0&audiochannels=lAudChannels&infochannels=0)

Get/SetAudioInputLevel

Get the current audio input level

Java: Must indicate the channel(s) to get/set. This is achieved by sending a long value representing the channel(s) you wish to get/set. For example if you wish to set audio channels 1 and 4 set the audChannels paramater to 9 (1001). You may also use the predefined audio channel defenitions in the MEDIACMD structure (audChanX, where x is the audio channel - 1).

Note: There are two get methods, one method will only return the current level.

The second method will return the current level as well as the max and min values for the audio channel in the GetValueMcmd object.

(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsAudInputLevel&position=lSetting &videochannels=0&audiochannels=lAudChannels&infochannels=0)

Get/SetAudioOutput

Get the current audio Output – See Get/SetAudioInput

Java: Must indicate the channel(s) to get/set. This is achieved by sending a long value representing the channel(s) you wish to get/set. For example if you wish to set audio channels 1 and 4 set the audChannels paramater to 9 (1001). You may also use the predefined audio channel defenitions in the MEDIACMD structure (audChanX, where x is the audio channel - 1).

Note: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the audio channel in the GetValueMcmd object.

(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsAudOutSelect&position=lSetting &videochannels=0&audiochannels=lAudChannels&infochannels=0)

Get/SetAudioOutputLevel

Get the current audio output level.

Java: Must indicate the channel(s) to get/set. This is achieved by sending a long value representing the channel(s) you wish to get/set. For example if you wish to set audio channels 1 and 4 set the audChannels paramater to 9 (1001). You may also use the predefined audio channel defenitions in the MEDIACMD structure (audChanX, where x is the audio channel - 1).

Note: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the audio channel in the GetValueMcmd object.

(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsAudOutputLevel&position=lSetting &videochannels=0&audiochannels=lAudChannels&infochannels=0)

GetAudioPeakRMS

Returns the RMS and Peak audio levels of the input (stop/record) or output (play/pause).

(XML: localhost/XMLMediaCmd?GetValue&cmdalt=gsAudWavePeakRMS)

Video Settings

Get/SetVideoInput

Get the current video input.

```
//! Standard NTSC or PAL composite video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE                    0x001
//! SVHS/S-Video four wire NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SVIDEO                       0x002
```

//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE_2 0x004
//! BetaCam level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV 0x010
//! Panasonic M2 level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_M2 0x020
//! SMPTE standard level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_SMPTE 0x040
//! RGB at video standard rate (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_RGB 0x080
//! D1 Serial Digital or HDSDI video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_D1_SERIAL 0x100
//! D1 Serial Parallel video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_D1_PARALLEL 0x200
//! SDTI/SDI including high speed transfer video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SDTI 0x400
//! No video available or no configurable settings (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_NONE 0
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidInSelect
&position=lSetting)

Get/SetVideoOutput

Get the current video output. See Get/SetVideoInput for settings.

Get/SetVideoInputSetup

Get the current video input's 'Setup' TBC setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Setup in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidInSetup&position=lSetting)

Get/SetVideoInputVideo

Get the current video input's 'Video' TBC setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Video in the GetValueMcmd object.

(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidInVideo
&position=lSetting)

Get/SetVideoInputHue

Get the current video input's 'Hue' TBC setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Hue in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidInHue
&position=lSetting)

Get/SetVideoInputChroma

Get the current video input's 'Chroma' TBC setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Chroma in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidInChroma
&position=lSetting)

Get/SetVideoTBCSetup

Get the current global TBC's 'Setup' setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Setup in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidSetup&position=lSetting)

Get./SetVideoTBCVideo

Get the current global TBC's 'Video' setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Videoin the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidVideo&position=lSetting)

Get/SetVideoTBCHue

Get the current global TBC's 'Hue' setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Hue in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidHue&position=lSetting)

Get/SetVideoTBCChroma

Get the current global TBC's 'Chroma' setting.
Java: There are two get methods, one method will only return the current level. The second method will return the current level as well as the max and min values for the Chroma in the GetValueMcmd object.
(XML: localhost/XMLMediaCmd?SetValue&cmdalt=gsVidChroma &position=lSetting)

Get/SetVideoGenlock

Turn the house/reference lock on or off

Get/SetVideoGenlockSource – not implemented externally

Set the source to input or external genlock

Error Log

vvwGetErrorLogMs

Get the ms time the last error was added to the error log

vvwSetErrorLog

Set the error log pointer to the message you want

vvwGetErrorLength

Get the length of the current error string

vvwGetError

Get the current error. Sets pointer to the next one automatically

System Settings

Get/SetCompressionRate

Get or set the current compression rate.

Get/SetSuperImpose – not implemented externally

Enable or disable the time code super impose on the main output.

GetTotalTime

Returns the total number of frames of storage available at current compression rate if the storage space was empty.

GetFreeTime

Returns the remaining number of frames of storage available at current compression rate.

GetTotalStorage

Returns the total storage connected in megabytes.

GetFreeStorage

Returns the amount of available storage for recording in megabytes.

GetCurMs

Get the current millisecond time from the controlled channel.

GetChannelCapabilities

Get the available commands for a channel.

GetVVWVersion

Returns the version string of the VVW subsystem.

GetMRVersion

Returns the version string of the MediaReactor subsystem.

GetVVWType

Returns the type string of the VVW channel.

Utility Functions

FreeString

ActiveX VB	*Not implemented*
ActiveX C++	*Not implemented*
Java	*Not implemented*
Dll	Void vvwFreeString (char * szString)
Unix	Void FreeString (char * szString)
XML	*Not supported*

Free a string value returned by the channel.

TCMaxFrame

ActiveX VB	vbx. TCMaxFrame (lFlags As Long) As Long
ActiveX C++	long pcx-> TCMaxFrame (long lFlags)
Java	long mci. TCMaxFrame (long lFlags)
Dll	*Use TCXlat*
Unix	Long TCMaxFrame (long lFlags)
XML	*Not supported*

Returns the maximum possible frame value for a time code type. See TCToFrame for flag definitions.

TCToFrame

ActiveX VB	vbx. TCToFrame (szTC As String, lFlags As Long) As Long
ActiveX C++	Long pcx-> TCToFrame (BSTR szTC, long lFlags)
Java	Long mci. TCToFrame (String szTC, long lFlags)
Dll	*Use TCXlat*
Unix	Long TCToFrame (char * szTC, long lFlags)
XML	*Not supported*

Convert a time code string to a frame count based on the flags.
Flags:

```
TC2_TCTYPE_MASK          0x000000FF
TC2_TCTYPE_FILM          0x00000001    // 24 fps
TC2_TCTYPE_NDF           0x00000002    // NTSC Non Drop Frame
TC2_TCTYPE_DF            0x00000004    // NTSC Drop Frame
TC2_TCTYPE_PAL           0x00000008    // PAL
TC2_TCTYPE_50            0x00000010    // PAL 720p (double rate)
TC2_TCTYPE_5994          0x00000020    // NTSC 59.94fps 720p
TC2_TCTYPE_60            0x00000040    // NTSC 60fps 720p
TC2_TCTYPE_NTSCFILM      0x00000080    // NTSC FILM 23.97

TC2_FTYPE_FIELD          0x10000000    // Field based (else frame)

// Basic sting tc representation types
TC2_STRTYPE_MASK         0x00000F00
TC2_STRTYPE_ASCII        0x00000100    // Std ascii string
TC2_STRTYPE_BCD          0x00000200    // RS-422 BCD or Packed
TC2_STRTYPE_HEX          0x00000400    // Hex packed DWORD
TC2_STRTYPE_GOP          0x00000800    // MPEG Gop TC
TC2_STRTYPE_INVERT       0x00001000    // Frames first

// Extended string handling
TC2_STREXT_MARKS         0x00010000    // Add : marks in string
TC2_STREXT_LEADING       0x00020000    // Include leading 0s
TC2_STREXT_TYPE          0x00040000    // Add ' N', ' D', ' P' or ' F' at end
TC2_STREXT_ALLCOLON      0x00080000    // No -.; just :
TC2_STREXT_FLAG          0x00100000    // Add DF Flag in BCD
TC2_STREXT_CF            0x00200000    // Add CF Flag in BCD
TC2_STREXT_MAX30         0x00400000    // Max 29 frames for output
TC2_STREXT_SHIFT7        0x40000000    // GOP Tc is shifted in DWORD
TC2_STREXT_SAVEBITS      0x80000000    // GOP Save unused bits
```

TCToString

ActiveX VB	vbx. TCToString (lTC As Long, lFlags As Long) As String
ActiveX C++	BSTR pcx-> TCToString (long lTC, long lFlags)
Java	String mci. TCToString (long lTC, long lFlags)
Dll	*Use TCXlat*
Unix	Char * TCToString (long lTC, long lFlags)
XML	*Not supported*

Convert a frame count to a time code string based on the flags. See TCToFrame for flag definitions.

VVWSpeedToPercentage

ActiveX VB	vbx. VVWSpeedToPercentage (lVVWSpeed As Long) As Double
ActiveX C++	Double pcx-> VVWSpeedToPercentage (long lVVWSpeed)
Java	Double mci. VVWSpeedToPercentage (long lVVWSpeed)
Dll	*Not implemented*
Unix	Double VVWSpeedToPercentage (long lVVWSpeed)
XML	*Not supported*

Convert a VVW speed (65520 based) to a percentage based speed (100.0).

PercentageToVVWSpeed

ActiveX VB	vbx. PercentageToVVWSpeed (double ddPercentageSpeed) As Long
ActiveX C++	Long pcx-> PercentageToVVWSpeed (double ddPercentageSpeed)
Java	Long mci. PercentageToVVWSpeed (double ddPercentageSpeed)
Dll	*Not implemented*
Unix	Long PercentageToVVWSpeed (double ddPercentageSpeed)
XML	*Not supported*

Convert a percentage speed (100.0) to a VVW speed (65520)

Java Data Structure Definitions

ClipInfo

Java	MediaCmdIF.ClipInfo clipData = new MediaCmdIF.ClipInfo(szClipName)

For use with the mci.GetClipInfo method. Must construct this data structure with a valid clip name.

Data members

String szClipName: clip name. This must be set to a valid clip name when a call is made to the mci.GetClipInfo method.
long lStart: the in point of the inquired clip.
long lEnd: the out point of the inquired clip.
long lVidEdit: number of video channels available for the requested clip.
long lAudEdit: number of audiochannels available for the inquired clip.
long lInfEdit: number of info channels available for the inquired clip.
String szFileName: file name of the requested clip.

VTREditLine

Java	MediaCmdIF.VTREditLine editInfo = new MediaCmdIF.VTREditLine();

For use with the mci.EDLGetEdit method. Values for the EDLGetEdit method are returned in the object's instance variables. The mci.EDLGetEdit method returns 0 for a successful retrieval and –1 on error. The VTREditLine instance variable will be set to null on error and will contain valid data on success.

Data members

long lRecordIn: time code on the VTR edit line.
long lPlayIn: the in point of the VTR edit line.
long lPlayOut: the out point of the VTR edit line.
long lVidEdit: number of video channels available for the VTR edit line.
long lAudEdit: number of audio channels available for the VTR edit line.
long lInfEdit: number of info channels available for the VTR edit line.
String szClipName: clip name of the current VTR edit line .
String szFileName: file name of the current VTR edit line.

GetValueMcmd

Java	MediaCmdIF.GetValueMcmd mCmdValues = new MediaCmdIF.GetValueMcmd(setValueType, pMin, pMax)

For use with the mci.ValueGet method. Must construct this data structure with a valid lValueType. If the min value of the get/set value is needed construct the GetValueMcmd instance with pMin different than –1. If the max value of the get/set value is needed construct the GetValueMcmd instance with pMax different than –1. Instantiating the GetValueMcmd with values of –1 for the pMin and/or pMax parameters will result in an invalid return for these instance variables.

Data members

long lValueType: the get/set value. Used by the mci.ValueGet method to determine which setting to retrieve.

long plMin: stores the min value for a get/set value if instantiated with a value different than –1.

long plMax: stores the max value for a get/set value if instantiated with a value different than –1.

AvailabePresets

Java	MediaCmdIF.AvailablePresets presets = new MediaCmdIF.AvaiablePresets()

For use with the mci.GetAvaolablePresets method. Values for the GetAvailablePresets method are returned in the object's instance variables.

Data members

long plVidEdit: supported video presets for a channel

long plAudEdit: supported audio presets for a channel

long plInfEdit: supported info presets for a channel

Appendix I – MediaCmd.h

```
/*****************************************************************************
*
 * $Id$:
 *
 * $HeadURL$:
 * $Author$:
 * $Revision$:
 * $Date$:
 *
 * Copyright © 1998-2007 Drastic Technologies Ltd.  All Rights Reserved.
 * 523 The Queensway, Suite 102 Toronto ON M8V 1Y7
 * 416 255 5636  fax 255 8780
 * engineering@drastictech.com      http://www.drastictech.com
 *****************************************************************************/

//
// Common header describing the command interface between
// media control modules.
//

#ifndef _MEDIACMD_INCLUDED_H
#define _MEDIACMD_INCLUDED_H

// Windows x64
#pragma warning(disable: 4996)          // deprecated security functions
#ifndef _CRT_SECURE_NO_WARNINGS
        #define _CRT_SECURE_NO_WARNINGS
#endif
#pragma warning( disable:4996 )

//#include <dlist.h>     // No longer used
//#ifdef _lint     // Test ONLY
//#include <windows.h>
//#endif

// This allows a quick check for the head of the command
//! Major command versioning for upgrades to the command set. see
MEDIACMD::dwCmdID
#define MEDIACMD_VERSION_MAJOR  0x0101UL
//! Minor command versioning for upgrades to the command set. see
MEDIACMD::dwCmdID
#define MEDIACMD_VERSION_MINOR  0x0003UL
//! Mask for checking the command set version. see MEDIACMD::dwCmdID
```

```
#define MEDIACMD_VERSION_MASK    0xFFFFUL
//! Permanent magic number of command id. see MEDIACMD::dwCmdID
#define MEDIACMD_CHECK_VER                0xFA250000UL
//! Mask for permanent magic number of command id. see MEDIACMD::dwCmdID
#define MEDIACMD_CHECK_MASK               0xFFFF0000UL
//! Current version and magic number.  Place in MEDIACMD::dwCmdID
#define MEDIACMD_CURRENT          (MEDIACMD_VERSION_MAJOR |
MEDIACMD_VERSION_MINOR | MEDIACMD_CHECK_VER)

// Various speed limits and definitions
//! Forward play speed (normal) in VVW (65520) see MEDIACMD::lSpeed
#define SPD_FWD_PLAY                      65520L
//! Pause speed (0%) in VVW (0) see MEDIACMD::lSpeed
#define SPD_PAUSE                         0L
//! Reverse play speed (-100%) in VVW (-65520) see MEDIACMD::lSpeed
#define SPD_REV_PLAY                      (-SPD_FWD_PLAY)
//! Maximum possible play speed in VVW see MEDIACMD::lSpeed
#define SPD_FWD_MAX                               5896800
//! Minimum possible play speed in VVW see MEDIACMD::lSpeed
#define SPD_REV_MAX                       (-SPD_FWD_MAX)
//! MAx speed for bumping
#define SPD_FAST_BUMP                     114660L //1.75 times play speed
//!Min Sped for buimping
#defineSPD_SLOW_BUMP                      32760l  //.5 times play speed
//! Illegal speed, set MEDIACMD::lSpeed to this value if not used
#define SPD_ILLEGAL                       2147483647L
//! Illegal time code reference, set MEDIACMD::dwPosition, MEDIACMD::dwStart,
MEDIACMD::dwEnd to this if not used
#define TC_ILLEGAL                        0xFFFFFFFFUL
//! Illegal channel, or All Channels.  Set MEDIACMD::dwAudioChannels,
MEDIACMD::dwVideoChannels, MEDIACMD::dwInfoChannels to this if not used
#define CHAN_ILLEGAL              0xFFFFFFFFUL
/**
* Maximum clip id length in the DEFAULT #MEDIACMD structure.  Larger versions may
be allocated and
* are legal.  Smaller versions should not have less than 10 bytes with zero padding
* for clip names (8 bytes + NULL) and file name (NULL).
* Note that the array is actually allocated as CMD_MAX_CLIP_ID_LEN+2+2
* 260+8+2+2 = 270 bytes (DWORD aligned)
* CLIP 8 Bytes
* NULL 1 Byte
* File 260 Bytes (_MAX_PATH)
* NULL 1 Byte
* 2 Alignment padding
*/
#define CMD_MAX_CLIP_ID_LEN               (260+8)

// The possible commands and states of media
/**
```

```
* The legal commands for a #MEDIACMD structure.  Set MEDIACMD::ctCmd to one of
* these values and expect it to be set to one of these values on a valid return
*/
enum cmdType {
        /**
        * Stop - Stop all playback, and normally place all channels into passthrough
        * <HR>
        */
        ctStop,                         // Stop all action
        /**
        * Pause - Halt all channels.  Display current video frame and silence audio<BR>
        * Seek - With cmdFlags::cfUsePosition and MEDICMD::dwPosition, goto that
frame and Pause
        * <HR>
        */
        ctPause,                        // Pause, Seek
        /**
        * Play - Play all channels.  May be modified by (cmdFlags::cfUseSpeed +
MEDIACMD::lSpeed) and
        * <BR> ((cmdFlags::cfUsePosition or cmdFlags::cfUsePositionOffset) and
MEDIACMD::dwPosition) or
        * <BR> ((cmdFlags::cfUseStart or cmdFlags::cfUseStartOffset) and
MEDIACMD::dwStart) or
        * <BR> ((cmdFlags::cfUseEnd or cmdFlags::cfUseEndOffset) and
MEDIACMD::dwEnd) as well as
        * <BR> MEDIACMD::dwCmdAlt with certain #cmdFlags
        * to play from-top, at speed or combinations of the above.
        * <HR>
        */
        ctPlay,                         // Play at specified speed (includes pause)
        /**
        * Record - Record one or a combination of video/audio/info to disk.  May be
modified, as with #ctPlay
        * by flags and structure members such as
        * <BR> ((cmdFlags::cfUsePosition or cmdFlags::cfUsePositionOffset) and
MEDIACMD::dwPosition) or
        * <BR> ((cmdFlags::cfUseStart or cmdFlags::cfUseStartOffset) and
MEDIACMD::dwStart) or
        * <BR> ((cmdFlags::cfUseEnd or cmdFlags::cfUseEndOffset) and
MEDIACMD::dwEnd) as well as
        * <BR> (cmdFlags::cfUseClipID and MEDIACMD::arbID) or
        * <BR> (cmdFlags::cfDeferred or cmdFlags::cfOverideDeferred) or
        * MEDIACMD::dwCmdAlt with certain #cmdFlags
        * to record from-top, at speed or combinations of the above.
        * <HR>
        */
        ctRecord,                       // Record at specified speed
        /**
```

```
        * Record Stop - Set the channel into a record ready state, normally passthrough
with the
        * recording file preallocated, and possible pass start end and name information.
See
        * <BR> (cmdFlags::cfUseStart cmdFlags::cfUseStartOffset and
MEDIACMD::dwStart)
        * <BR> (cmdFlags::cfUseEnd cmdFlags::cfUseEndOffset and
MEDIACMD::dwEnd)
        * <BR> (cmdFlags::cfUseClipID and MEDIACMD::arbID)
        * for record setups.
        * <HR>
        */
        ctRecStop,                      // Stop ready for recording
        /**
        * Eject - Stop the channel and unload removable media, if possible, else same as
stop
        * <HR>
        */
        ctEject,                // Eject the current media
        /**
        * Transfer - Transfer media from one channel to another.  Normally used to
transfer
        * internal media to or from an external tape device.
        * <HR>
        */
        ctTransfer,                     // Transfer source from one channel to another
        /**
        * Insert Clip or Timecode Area - Used in time code space (TCSpace.h) and Clip
Space
        * (ClipSpace.h) to add new clips or areas.  Inserted media is defined by
        * (cmdFlags::cfUseStart - MEDIACMD::dwStart, cmdFlags::cfUseEnd -
MEDIACMD::dwEnd) for
        * clip being added and cmdFlags::cfUsePosition - MEDIACMD::dwPosition for
target.  Also
        * (cmdFlags::cfUseClipID and MEDIACMD::arbID) may be used to specify a file
name.
        * cmdFlags::cfUsePresets and MEDIACMD::dwVideoChannels,
MEDIACMD::dwAudioChannels,
        * MEDIACMD::dwInfoChannels are also respected if set.
        * cmdFlags::cfRipple may also be used to insert over
        * <BR> NOTE - The ctTransfer command is ALWAYS sent to the target with the
SOURCE channel
        * in the MEDIACMD::dwCmdAlt member and cmdFlags::cfUseCmdAlt set UNLESS
one of the
        * devices is slow/high latency/sloppy (read VTR), in which case it always recieves
the
        * command so it can master the transfer and the cmdFlags::cfInvert is used to
set the direction.
        * <HR>
```

```
        */
        ctInsert,                       // Insert a new time code area
        /**
        * Blank a Timecode Area - Used in time code space (TCSpace.h) to set an area
to black
        * and silent audio.
        * (cmdFlags::cfUseStart - MEDIACMD::dwStart, cmdFlags::cfUseEnd -
MEDIACMD::dwEnd) set
        * the area to be blanked.
        * cmdFlags::cfUsePresets and MEDIACMD::dwVideoChannels,
MEDIACMD::dwAudioChannels,
        * MEDIACMD::dwInfoChannels are also respected if set.
        * cmdFlags::cfRipple may also be used to remove blank area.  With this
        * command, no media is removed from storage.
        * <HR>
        */
        ctBlank,                // Erase the specified TC area
        /**
        * Delete a clip (ClipSpace.h) or an area (TCSpace.h).
        * Deletes the media from storage and from the current space.
        * For ClipSpace, cmdFlags::cfUseClipID and MEDIACMD::arbID must be
specified, and
        * if any sub clip or super clips exist, the id will be removed but the media
        * will not be deleted.
        * For TCSpace, cmdFlags::cfUseStart and MEDIACMD::dwStart with
cmdFlags::cfUseEnd and
        * MEDIACMD::dwEnd should be used to specify the time code segment to be
deleted.
        * cmdFlags::cfUsePresets and MEDIACMD::dwVideoChannels,
MEDIACMD::dwAudioChannels,
        * MEDIACMD::dwInfoChannels may also be used to delete specific channels.
        * If cmdFlags::cfRipple is set, then the TCSpace will close over the deleted
        * material, changing all timecode location beyond the deletion point by minus the
        * size of the deletion.
        * <HR>
        */
        ctDelete,                       // Slide segment within the specified TC area
        /**
        * Trim a clip or area - Currently not implemented.  Use cmdType::ctSetValue and
#cmdGetSetValue::gsClipInfo
        * to trim a clip, or a combination of cmdType::ctInsert, cmdType::ctDelete,
cmdType::ctBlank
        * to trim a tcspace area.
        * <HR>
        */
        ctTrim,                         // Trim the segment within the specified TC area
        /**
        * Channel select - select active channels, preview passthrough channels (to
preview and edit)
```

```
        * recording channels (to create a split edit ala CMX)
        * Requires cmdFlags::cfUsePresets and MEDIACMD::dwVideoChannels,
MEDIACMD::dwAudioChannels,
        * MEDIACMD::dwInfoChannels
        * <HR>
        */
        ctChanSelect,           // Pass though requested channels

        /**
        * Get the current state of the controlled channel(s) - Fills the user supplied
        * #MEDIACMD structure with the current state.  Look for
        * cmdType::ctError, cmdType::ctStop, cmdType::ctEject, cmdType::ctPause,
cmdType::ctPlay,
        * cmdType::ctRecStop, cmdType::ctRecord for basic state.  For valid fields,
check
        * <ul>
        * <li>cmdFlags::cfDeferred : we have a deferred clip
        * <li>cmdFlags::cfTimeMs : MEDIACMD::dwCmdAlt has millisecond performance
counter info
        * <li>cmdFlags::cfUseSpeed : MEDIACMD::lSpeed has the valid current speed
        * <li>cmdFlags::cfUsePresets : MEDIACMD::dwVideoChannels,
MEDIACMD::dwAudioChannels, MEDIACMD::dwInfoChannels contain preset information
        * <li>cmdFlags::cfUsePosition : MEDIACMD::dwPosition contains current
position
        * <li>cmdFlags::cfUseStart : MEDIACMD::dwStart has starting frame position
        * <li>cmdFlags::cfUseEnd : MEDIACMD::dwEnd has end frame postion (+1 the
out is never included)
        * <li>cmdFlags::cfUseClipID : MEDIACMD::arbID has current clip name (8 char
for louth and odetics)
        * <li>cmdFlags::cfFields : MEDIACMD::dwPosition, MEDIACMD::dwStart and
MEDIACMD::dwEnd are in fields if they are valid
        * <li>cmdFlags::cfNoReturn : return is invalid.
        * </ul>
        * <HR>
        */
        ctGetState,                     // Returns TC and transport state information
        /**
        * Set the current state - Used for control type channels such as Serial 422 control
(vvwCtl.h) and
        * network control (vvwNet.h).  Tells the controller or user what our current state
is.  The state
        * should be reported honestly, as it is the receivers responsibility to transition
states in
        * an appropriate way for its controller.  For actual channels (vvwInt.h, vvwExt.h,
vvwNet.h-as controller,
        * vvwDS2.h, etc), the state should be set by using one of the transport
commands (cmdType::ctPlay etc) above.
        * <HR>
        */
```

```
        ctSetState,                     // Sends a new state per GetState
        /**
        * Get a non transport setting - Used for one time setups on channel.
        * Includes audio levels, video proc amps, audio/video input, compression type
and level and
        * many others
        * See:  #cmdGetSetValue for possible commands
        * <HR>
        */
        ctGetValue,                     // Get value of video, audio of internal variable
        /**
        * Set a non transport setting - Used for one time setups on channel.
        * Includes audio levels, video proc amps, audio/video input, compression type
and level and
        * many others
        * See:  #cmdGetSetValue for possible commands
        * <HR>
        */
        ctSetValue,                     // Set value of video, audio of internal variable
        /**
        * Check support for a non transport setting - Used for one time setups on
channel.
        * Includes audio levels, video proc amps, audio/video input, compression type
and level and
        * many others
        * See:  #cmdGetSetValue for possible commands
        * <HR>
        */
        ctValueSupported,       // Returns true is the specified Get/Set value is
supported
        /**
        * Indicates the an error in the channel has occurred.  Return only.
        * See MEDIACMD::dwCmdAlt for error code and MEDIACMD::arbID for message
if any.
        * These members will be valid if cmdFlags::cfUseCmdAlt and
cmdFlags::cfUseClipID are set
        * <HR>
        */
        ctError,                    // An error has occurred
        /**
        * Terminate Close A Channel - Only used by remote devices that cannot close
the channel
        * directly such as vvwNet.h.  Channel may not actually close when this is called,
but
        * the communications pipe will be closed and wait for another connection.
        * <HR>
        */
        ctTerminate,            // Terminate the current command and move to the next
in the queue, if any
```

```
        /**
        * Abort the current operation - Use to abort operations that would normally
ignore
        * extraneous commands such as non-linear playback sequences, records or if the
        * channel just seems to be stuck.  Makes a good panic button.
        * <HR>
        */
        ctAbort                         // Abort any queued commands
};

/**
* Flags that modify #cmdType in the #MEDIACMD structure.  Mostly used to specify
* which fields in the structure are valid.
*/
enum cmdFlags {
                // Normally, commands occur when the delay time is reached
        /**
        * Delay this command until the end of the previous one.  This is the method for
playing
        * back clips non-linearly.  Send one clip to play, then send each clip after it with
        * this flag set and they will play seemlessly back to back.<BR>
        * In the case of ctInsert, the deferred indicates that the clip to be inserted
        * will be translated from its current location to the current record directory
        * and then added to the bin/tcspace.
        * <HR>
        */
        cfDeferred = 1,                         // 0x00000001 This is a delayed
command (either at end of prev cmd, or absolute time)
        /**
        * Delay the command, as in #cfDeferred, but kill any other waiting commands
and use
        * this command as soon as the current command completes.
        * <HR>
        */
        cfOverideDeferred = 1 << 30,// 0x40000000 Overide all previous deferred
commands
        /**
        * Time is in milliseconds.  Applies only to the MEDIACMD::dwCmdAlt member.
The
        * millisecond reference is derived from the performance counter (or one
extremely
        * old machines timeGetTime()) via vsyncGetCurMs() which is implemented in
DSync.dll
        * for user and kernal modes. The defualt timing without this flags set is in video
frames.
        * <HR>
        */
        cfTimeMs = 1 << 1,                      // 0x00000002 Use Millisecond time for
delayed time, not fields
```

```
        /**
        * Time is set for event occurance.  This means the command will occur when the
time
        * specified is reached.  If this flag is not set and #cfTimeMs is set, then the time
        * indicates the time the command was recieved and may be used for a
deterministic
        * offset.  May be in frames (default) or milliseconds #cfTimeMs, requires
#cfUseCmdAlt.
        * <HR>
        */
        cfTimeTarget = 1 << 2,                  // 0x00000004 Delayed time is offset
from current time code
        /**
        * Time reference is the system clock (time of day) not the performance clock.
This is
        * used to sync network or serial based communication where there is no
relationship
        * between performance clocks.  For proper operation, the two devices must be
genlocked
        * to the same video source, which VVW will interpolate with the correcte system
clock
        * to keep everything together.  Note:  This is only as accurate as the genlock
readers
        * an LTC or Network time transport connected to BOTH machines.  In general, a
pair
        * of VVWs are accurate to 1 field, which is ample for editing and broadcast
insertion
        * <HR>
        */
        cfTimeHouseClock = 1 << 3,    // 0x00000008 Delayed time is based on
absolute (real) time
        /**
        * Means the MEDIACMD::lSpeed member is valid.
        * <HR>
        */
        cfUseSpeed = 1 << 4,            // 0x00000010 Set the new speed
        /**
        * Means the MEDIACMD::dwVideoChannels, MEDIACMD::dwAudioChannels and
MEDIACMD::dwInfoChannels members are valid.
        * <HR>
        */
        cfUsePresets = 1 << 5,          // 0x00000020 Use video and audio edit presets
        /**
        * Means the MEDIACMD::dwPosition member is valid.
        * <HR>
        */
        cfUsePosition = 1 << 6,                 // 0x00000040 Use the position setting
        /**
```

```
        * Means the MEDIACMD::dwPosition member is valid and should be used as a
long (signed)
        * against the current channel position counter.
        * <HR>
        */
        cfUsePositionOffset = 1 << 7,//0x00000080 Position is an offset
        /**
        * Means the MEDIACMD::dwStart member is valid.
        * <HR>
        */
        cfUseStart = 1 << 8,            // 0x00000100 Start a new timecode
        /**
        * Means the MEDIACMD::dwStart member is valid and should be used as a long
(signed)
        * against the current channel position counter.
        * <HR>
        */
        cfUseStartOffset = 1 << 9,      // 0x00000200 Start is an offset from current tc
        /**
        * Means the MEDIACMD::dwEnd member is valid.
        * <HR>
        */
        cfUseEnd = 1 << 10,                     // 0x00000400 End command as
specified
        /**
        * Means the MEDIACMD::dwEnd member is valid and should be used as a long
(signed)
        * against the current channel position counter.
        * <HR>
        */
        cfUseEndOffset =  1 << 11,     // 0x00000800 End is and offset from current tc

        /**
        * Causes the command to act on all IDs in the system.  Used for clipspace to
        * delete all ids quickly.
        * <HR>
        */
        cfUseAllIDs = 1 << 12,          // 0x00001000 Use all clip IDs (usually erase'em)
        /**
        * Means the MEDIACMD::arbID member is valid.
        * <HR>
        */
        cfUseClipID = 1 << 13,          // 0x00002000 Use new clip ID, otherwise use
last or none
        /**
        * Copy the media to the current record folder when inserting
        * <HR>
        */
        cfCopy = 1 << 14,       // 0x00004000
```

```
        /**
        * Means the command should not be used on any clip or clip spaces
        * <HR>
        */
        cfNoClipFiles = cfCopy, // Deprecated
        /**
        * Convert the media to the current record folder when inserting
        * <HR>
        */
        cfConvert = 1 << 15,   // 0x00008000
        /**
        * Means the command should not be used on any clip within or the TCSpace
itself
        * <HR>
        */
        cfNoTCSpaces = cfConvert,               // Deprecated
        /**
        * Means the MEDIACMD::dwCmdAlt is valid
        * <HR>
        */
        cfUseCmdAlt = 1 << 16,                  // 0x00010000 Use the dwCmdAlt
        /**
        * Sent by shuttle/jog/var controllers for drivers that require
        * a special play state that take too much time to get into.  If
        * this flag is true, the command is a shuttle and true play
        * does not need to be used
        * <HR>
        */
        cfIsShuttle = 1 << 17,          // 0x00020000 Use speed in play for shuttle
        /**
        * If set then elements that are not illegal are current at the reception of the
command
        * <BR>If #cfUsePosition and #cfUsePositionOffset are NOT set and
#MEDIACMD::dwPosition is not #TC_ILLEGAL,
        * then it is the current position when the command was recieved
        * <BR>If #cfUseStart and #cfUseStartOffset are NOT set and
#MEDIACMD::dwStart is not #TC_ILLEGAL,
        * then it is the current start location when the command was recieved
        * <BR>If #cfUseEnd and #cfUseEndOffset are NOT set and
#MEDIACMD::dwEnd is not #TC_ILLEGAL,
        * then it is the current end time when the command was recieved
        * <BR>If #cfUseSpeed is set are NOT set and #MEDIACMD::lSpeed is not
#SPD_ILLEGAL,
        * then it is the current speed when the command was recieved
        * <BR>If cfUsePresets is NOT set and #MEDIACMD::dwAudioChannels,
#MEDIACMD::dwVideoChannels and #MEDIACMD::dwInfoChannels are not 0xFFFFFFFF,
        * then they are the current presets when the command was recieved
        * <BR>If #cfUseClipID is NOT set and #MEDIACMD::arbID[0] is not equal to
NULL (""),
```

```
        * then it is the current clip id when the command was recieved.  If
#MEDIACMD::arbID[9] is not equal to NULL ("")
        * then it is the current file name when the command was recieved.
        * <HR>
        */
        cfUsingCurrent = 1 << 18,       // 0x000400000 any elem not flag'ed is current
        /**
        * If set then MEDIACMD::dwPosition, MEDIACMD::dwStart and
MEDIACMD::dwEnd are absolute
        * (0 based) frame counts, else they are the current type (CTL/CLIP(frame count)
of
        * LTC/VITC(time code offset)).
        * <HR>
        */
        cfUseFrameCount = 1 << 19,   // 0x000800000 Position, start and end are
fields, not frames
        /**
        * If set then MEDIACMD::dwPosition, MEDIACMD::dwStart and
MEDIACMD::dwEnd should be
        * interpreted as fields, not frames, if they are valid
        * <HR>
        */
        cfFields = 1 << 20,                     // 0x00100000 Position, start and end
are fields, not frames
        /**
        * Close up any holes created by this command.  Most importantly
cmdType::ctDelete,
        * cmdType::ctBlank, cmdType::ctInsert and cmdType::ctTrim.
        * <HR>
        */
        cfRipple = 1 << 21,                     // 0x00200000 Ripple for insert or delete
        /**
        * Command should be looped.  Mostly used for loop playback where an start and
end are
        * specified.  The play will begin at the start, proceed to the end, and once
reached
        * loop back to the start again.
        * <HR>
        */
        cfLoop = 1 << 22,                       // 0x00400000 Loop the clip or in out
        /**
        * INTERNAL - Allows one channel to setup a DSync trigger with another.  Use
cmdType::ctTransfer
        * instead as this is very inefficient for non local command transports.
        * <HR>
        */
        cfTrigger = 1 << 23,            // 0x00800000 Trigger using dsync class
        /**
```

```
        * This command is part of a preview.  Either it notes a channel change (pass
through to emulate
        * an edit) or that the playback does not have to be consistant and frame
accurate.  Also
        * returned if the channel can only produce preview quality playback (eg. VGA
playback
        * of HDTV media without hardware assist).
        * <HR>
        */
        cfPreview = 1 << 24,            // 0x01000000 Preview set (EE, non rt play)
        /**
        * This tells the ddr that the command originated from a remote machine before
being
        * accepted from vvwNet. This is the only way to tell if we have full system acces
        * if this flag is set, windows commands (HANDLES) will be ignored at the avHal
Level
        * this is for all commands not origanating from localhost
        *
        *
        * <HR>
        */
        cfRemoteCommand = 1 << 25,
        /**
        * When returned in a status, it means the second field in time (the later field) is
        * the current one being display.  When sent, it indicates which field is to be
displayed
        * if only on field is going to be displayed, or which field to start the edit on (edit
        * start is NOT supported in the 3.0 version of VVW).
        * <HR>
        */
        cfSecondField = 1 << 27,                // 0x08000000 Is/Use second field
temporally
        /**
        /*****************************************************************
*******/
        /* When used in pause command it will advance to the next field
*/
        /*****************************************************************
*******/
        cfUseNextField = cfSecondField,
        /**
        * For cmdType::ctTransfer, invert the source and target.  Use to allow an
external device (such
        * as a VTR) to always master the transfer procedure.  Because of the high
latency and poor
        * ballistics of all VTRs, the internal transfer slaves to it regardless of whether it is
        * is the source of target of the transfer.
        * <HR>
        */
```

```
        cfInvert = 1 << 28,                     // 0x10000000 Invert a transfer
        /**
        * Means do not act on this command, but return #GS_NOT_SUPPORTED in
dwPosition if you cannot
        * handle it.  Used to determine basic capabilities of the channel.  For instance, if
        * its an MPEG2 playback channel, it can't record but if it has a passthrough, it
may be
        * able to stop.  Using cfTest with cmdType::ctRecord, cmdType::ctStop will tell
the
        * caller this so the interface may be adjusted accordingly.
        * <BR> Caution:  This flag has not been tested with all transport types.  Avoid
for now.
        * <HR>
        */
        cfTest = 1 << 29,                       // 0x20000000 See if the command
exists
        // NOTE:  1 << 30 in use by cfOverideDeferred
        /**
        * Instucts the channel that no return is required.  The channel then has the
option of
        * remembering the command and acting on it within a reasonable time.  This
means the
        * caller does not know if the command completed successfully at return time, but
the
        * status should be monitored anyways to figure that out.  Especially when long
time
        * functions like a VTR seek will return that the command was successfully
initiated,
        * but not wait for the completion of the seek, regardless of this flag.
        * <HR>
        */
        cfNoReturn = 1UL << 31UL                // 0x80000000 No return mediacmd is
required
};

// Video channels
//! Video channel bit array for MEDIACMD::dwVideoChannels
//! @{
enum cmdVidChan {
        vidChan0 = 1, vidChan1 = 1 << 1, vidChan2 = 1 << 2, vidChan3 = 1 << 3,
        vidChan4 = 1 << 4, vidChan5 = 1 << 5, vidChan6 = 1 << 6, vidChan7 = 1 <<
7,
        vidChan8 = 1 << 8, vidChan9 = 1 << 9, vidChan10 = 1 << 10, vidChan11 = 1
<< 11,
        vidChan12 = 1 << 12, vidChan13 = 1 << 13, vidChan14 = 1 << 14, vidChan15
= 1 << 15,
        vidChan16 = 1 << 16, vidChan17 = 1 << 17, vidChan18 = 1 << 18, vidChan19
= 1 << 19,
```

```
        vidChan20 = 1 << 20, vidChan21 = 1 << 21, vidChan22 = 1 << 22, vidChan23
= 1 << 23,
        vidChan24 = 1 << 24, vidChan25 = 1 << 25, vidChan26 = 1 << 26, vidChan27
= 1 << 27,
        vidChan28 = 1 << 28, vidChan29 = 1 << 29, vidChan30 = 1 << 30, vidChan31
= 1UL << 31UL,
#define vidChanAll 0xFFFFFFFFUL
};
//! @}

//! Audio channel bit array for MEDIACMD::dwAudioChannels
//! @{
enum cmdAudChan {
        audChan0 = 1, audChan1 = 1 << 1, audChan2 = 1 << 2, audChan3 = 1 << 3,
        audChan4 = 1 << 4, audChan5 = 1 << 5, audChan6 = 1 << 6, audChan7 = 1
<< 7,
        audChan8 = 1 << 8, audChan9 = 1 << 9, audChan10 = 1 << 10, audChan11 =
1 << 11,
        audChan12 = 1 << 12, audChan13 = 1 << 13, audChan14 = 1 << 14,
audChan15 = 1 << 15,
        audChan16 = 1 << 16, audChan17 = 1 << 17, audChan18 = 1 << 18,
audChan19 = 1 << 19,
        audChan20 = 1 << 20, audChan21 = 1 << 21, audChan22 = 1 << 22,
audChan23 = 1 << 23,
        audChan24 = 1 << 24, audChan25 = 1 << 25, audChan26 = 1 << 26,
audChan27 = 1 << 27,
        audChan28 = 1 << 28, audChan29 = 1 << 29, audChan30 = 1 << 30,
audChan31 = 1UL << 31UL,
#define audChanAll 0xFFFFFFFFUL
};
//! @}

//! Info channel bit array for MEDIACMD::dwInfoChannels
//! @}
enum cmdinf {
        //! LTC time code user bit channel
        infLtc = 1,
        //! VITC time code user bit channel
        infVitc = 1 << 1,
        //! Incoming source control time code
        infSrcCtl = 1 << 2,
        //! Incoming source LTC time code user bits
        infSrcLtc = 1 << 3,
        //! Incoming source VITC time code user bits
        infSrcVitc = 1 << 4,
        //! Record time of day
        infRecTime = 1 << 5,
        //! Record Data
        infRecDate = 1 << 6,
```

```
        //! Close caption infomation
        infCC = 1 << 7,
        //! Authorization information
        infAuth = 1 << 8,
        //! Copyright information
        infCopyright = 1 << 9,
        //! Ownership information
        infOwner = 1 << 10,
        //! Source media name
        infSourceName = 1 << 11,
        //! Source proxy name (if any)
        infProxyName = 1 << 12,
        //! Unused inf13 - inf21
        inf13 = 1 << 13, inf14 = 1 << 14, inf15 = 1 << 15,
        inf16 = 1 << 16, inf17 = 1 << 17, inf18 = 1 << 18, inf19 = 1 << 19,
        inf20 = 1 << 20, inf21 = 1 << 21, infVB0 = 1 << 22, infVB1 = 1 << 23,
        infVB2 = 1 << 24, infVB3 = 1 << 25, infVB4 = 1 << 26, infVB5 = 1 << 27,
        infVB6 = 1 << 28, infVB7 = 1 << 29, infVB8 = 1 << 30, infVB9 = 1UL <<
31UL,
#define infChanAll 0xFFFFFFFFUL
};

//! @{
/**
* Enum sent in MEDIACMD::dwCmdAlt for the commands cmdType::ctGetValue,
* cmdType::ctSetValue, cmdType::ctValueSupported.  <BR>
* ctGetValue will return information in the mediacmd per this
* document <BR>
* ctSetValue will change the state of the channel using the
* members of mediacmd per this document <BR>
* ctValueSupport will return GS_NOT_SUPPORTED in #MEDIACMD::dwPosition
* if it NOT supported.  If it is supported, #MEDIACMD::dwPosition will
* be set to some other value <BR>
* <HR>
* NOTE:  <BR>
* \li 'nada' is spanish for nothing and is used here to indicate
* that the command is not supported.
* \li  Time Code - There are three main time code types, each with their
* own user bit information.  The 0 based absolute time code is refered to
* by 'Tc' and 'Ub'.  The LTC (longitudinal time code or SMPTE time code
* often sent via audio) is refered to by 'LtcTc' and 'LtcUb'.  The VITC
* (vertical interval time code, usually encoded in the vertical blank
* area of the video signal) is refered to by 'VitcTc' and VitcUb'.  Not
* all devices will support all types or the user bits value for some
* types.  Use value supported to determin support
* <HR>
*/
enum cmdGetSetValue {
        /**
```

```
* Current internal time - control or clip absolute zero based
* time code (0..total frames exclusive)
* \li cmdType::ctSetValue
* <BR> nada
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition
* <HR>
*/
gsTc = 1,                                   // Current internal TC (dwPosition)
/**
* Current internal user bits
* <BR>
* \li cmdType::ctSetValue
* <BR>  nada
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition
* <HR>
*/
gsUb,                                       // Current user bits (dwPosition)
/**
* Current LTC time
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set vitc generator for
* next gen (record) if in preset mod
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition
* <HR>
*/
gsLtcTc,                                    // Current LTC TC (dwPosition)
/**
* Current LTC user bits
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set ltc generator
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition
* <HR>
*/
gsLtcUb,                                    // Current LTC user bits (dwPosition)
/**
* Current VITC time
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set ltc generator
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition
* <HR>
*/
```

```
        gsVitcTc,                       // Current VITC TC (dwPosition)
        /**
        * Current VITC user bits
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - to set vitc generator
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition
        * <HR>
        */
        gsVitcUb,                       // Current VITC user bits (dwPosition)
        /**
        * Current time code source
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - #GS_TCSOURCE_LTC,
#GS_TCSOURCE_VITC, #GS_TCSOURCE_CTL or #GS_TCSOURCE_CLIP
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - #GS_TCSOURCE_LTC,
#GS_TCSOURCE_VITC, #GS_TCSOURCE_CTL or #GS_TCSOURCE_CLIP
        * <BR>  MEDIACMD::dwStart - supported types using bit array of above
        * <HR>
        */
        gsTcSource,                     // Default Source (dwPosition, supported
dwStart)
        /**
        * Current time code type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - #TC2_TCTYPE_FILM, #TC2_TCTYPE_NDF,
#TC2_TCTYPE_DF, #TC2_TCTYPE_PAL, #TC2_TCTYPE_50, #TC2_TCTYPE_5994,
#TC2_TCTYPE_60, #TC2_TCTYPE_NTSCFILM, #TC2_TCTYPE_2398, #TC2_TCTYPE_100
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - #TC2_TCTYPE_FILM, #TC2_TCTYPE_NDF,
#TC2_TCTYPE_DF, #TC2_TCTYPE_PAL, #TC2_TCTYPE_50, #TC2_TCTYPE_5994,
#TC2_TCTYPE_60, #TC2_TCTYPE_NTSCFILM, #TC2_TCTYPE_2398, #TC2_TCTYPE_100
        * <BR>  MEDIACMD::dwStart - supported types using bit array of above
        * <HR>
        */
        gsTcType,                       // DF, NDF, PAL or FILM
(dwPosition,supported dwStart)
        /**
        * Lowest possible time code frame
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - New minimum value
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current minimum value
        * <BR>  MEDIACMD::dwStart - Absolute minimum possible value (usually 0)
```

```
        * <HR>
        */
        gsStart,                        // Lowest possible TC (current = dwPosition, min
= dwStart)
        /**
        * Highest possible time code frame plus 1 (out is never included)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - New maximum value + 1
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current maximum value + 1
        * <BR>  MEDIACMD::dwStart - Absolute maximum possible value (usually clip
end + 1)
        * <HR>
        */
        gsEnd,                          // Highest possible TC (current =
dwPosition, max = dwStart)
        /**
        * Current mark in time set by any caller
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - New in time
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Current in time
        * <HR>
        */
        gsIn,                           // Current in point (dwPosition)
        /**
        * Previous (currently non active)  mark in time set by RS-422 protocol
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported (internal)
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - last known in time
        * <HR>
        */
        gsLastIn,                       // Last in point (dwPosition)
        /**
        * Current mark out time set by any caller
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - New out time
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Current out time
        * <HR>
        */
        gsOut,                          // Current out point (dwPosition)
        /**
        * Previous (currently non active) mark out time set by RS-422 protocol
```

```
* <BR>
* \li cmdType::ctSetValue
* <BR>  - not supported (internal)
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - last known out time
* <HR>
*/
gsLastOut,                              // Last out point (dwPosition)
/**
* Number of frames from edit on command to start of record (usually 4~7)
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - New number of frames
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current number of frames
* <HR>
*/
gsEditOn,                               // Time to start an edit
/**
* Number of frames from edit off command to end of record (usually 4~7)
* should match #gsEditOn in most cases
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - New number of frames
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current number of frames
* <HR>
*/
gsEditOff,                              // Time to end an edit
/**
* Number of frames to preroll before in point for an edit
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - New number of frames
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current number of frames
* <HR>
*/
gsPreroll,                              // Edit pre roll time
/**
* Number of frames to postroll after an out point for an edit
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - New number of frames
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current number of frames
* <HR>
*/
gsPostroll,                             // Edit post roll time
```

```
/**
* Switch from normal mode to auto mode.  For sony vtr emulation
* it sets up Pioneed dual head emulation.  For Louth and Odetics
* enables preview play look ahead for seemless clip playback
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - #GS_TRUE or #GS_FALSE
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current auto mode state as above
* <HR>
*/
gsAutoMode,                                  // Setup for NL Playback
/**
* Number of frames from recieving play command to actual play
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - New number of frames
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - Current number of frames
* <HR>
*/
gsPlayDelay,                         // Time from pause to play
/**
* LTC time code preset (generator preset)
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set generator for the next
* next record.  Will be used not in regen mode.
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - returns the current generator
* preset.
* <HR>
*/
gsLtcTcPreset,
/**
* LTC user bit preset (generator preset)
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set generator for the next
* next record.  Will be used not in regen mode.
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - returns the current generator
* preset.
* <HR>
*/
gsLtcUbPreset,
/**
* VITC time code preset (generator preset)
```

```
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set generator for the next
* next record.  Will be used not in regen mode.
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - returns the current generator
* preset.
* <HR>
*/
gsVitcTcPreset,
/**
* VITC time code preset (generator preset)
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwPosition - to set generator for the next
* next record.  Will be used not in regen mode.
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwPosition - returns the current generator
* preset.
* <HR>
*/
gsVitcUbPreset,
/**
* Returns the block of data for a frame
* <BR>
* \li cmdType::ctSetValue
* <BR>  Not really used at this point
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::arbID - The data block
* <BR>  MEDIACMD::dwEnd - The size of the data block (0..n)
* <BR>  MEDIACMD::dwPosition - The expected type of data
* <HR>
*/
gsFrameData,
/**
* Current Key Code
* <BR>
* \li cmdType::ctSetValue
* <BR>  MEDIACMD::dwStart - Key Code prefix (4 bytes)
* <BR>  MEDIACMD::dwPosition - Key Code (4 bytes)
* \li cmdType::ctGetValue
* <BR>  MEDIACMD::dwStart - Key Code prefix (4 bytes)
* <BR>  MEDIACMD::dwPosition - Key Code (4 bytes)
* <HR>
*/
gsKeyCode,                              // Key Code
/**
* Current Ink Code
* <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwStart - Ink Code prefix (4 bytes)
        * <BR>  MEDIACMD::dwPosition - Ink Code (3 bytes)
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwStart - Ink Code prefix (4 bytes)
        * <BR>  MEDIACMD::dwPosition - Ink Code (3 bytes)
        * <HR>
        */
        gsInkCode,                          // Ink Code
        /**
        * Current 215 Code Code
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwStart - Audio Phase [msb], Audio Modulus, pull down,
sequence [lsb]
        * <BR>  MEDIACMD::dwPosition - Absolute Frame
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwStart - Audio Phase [msb], Audio Modulus, pull down,
sequence [lsb]
        * <BR>  MEDIACMD::dwPosition - Absolute Frame
        * <HR>
        */
        gs215Code,                          // Other 215 Codes
        /**
        * Set the heads and tails to be used for the next record.  Automatically set to
zero after record/add
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - Heads and Tails size in frames
        * \li cmdType::ctGetValue - not supported
        * <BR>
        * <HR>
        */
        gsHeadsAndTails,                // Number of frames being added for heads and
tails



        gsGetNextClip = 90,             // Get clip name and info (was first/next - send
NULL arbID for first)
        /**
        * Obsolete - use #gsGetNextClip
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported in new drivers
        * \li cmdType::ctGetValue
        * <BR>  - not supported in new drivers
        * <HR>
        */
```

```
        gsFirstClip,                    // First clip name (arbID - name, dwStart, dwEnd
if avail)
        /**
        * Obsolete - use #gsGetNextClip
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported in new drivers
        * \li cmdType::ctGetValue
        * <BR>  - not supported in new drivers
        * <HR>
        */
        gsNextClip,                     // Next clip name, (arbID - name,
dwStart, dwEnd if avail)
        /**
        * Return the next state info when working through a time code space time line
to
        * retrieve all the edits in order.  The state uses MEDIACMD::dwPosition,
        * MEDIACMD::dwVideoChannels, MEDIACMD::dwAudioChannels,
MEDIACMD::dwInfoChannels
        * to maintain the state (Please note that MEDIACMD::arbID is reserved and must
        * be maintained between calls).  The dwPosition describes the current position
        * in the time line and the channel bits are set for channels already returned.
        *  See gsTCSGetTLNextClip for more info
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Current time line position
        * <BR>MEDIACMD::dwVideoChannels - Video channels used so far
        * <BR>MEDIACMD::dwAudioChannels - Audio channels used so far
        * <BR>MEDIACMD::dwInfoChannels - Info channels used so far
        * <HR>
        */
        gsTCSGetTLClipState,
        /**
        * CALL                          Pos         Start   End         V
              A           I             arbID
        * gsTCSGetTLClipInfo  0         x           x           0
        0           0           x           - Restart at 0
        *       Rtn                             0           0
        300         1           2           0           file1   - 10 sec
VA2 from file1
        * gsTCSGetTLNextState 0                                       0
        0           0                       - First state 0
        *       Rtn                         0
                1           2           0                       - First
clip channels
        *       ( Copy prev gsTCSGetTLNextState into gsTCSGetTLClipInfo before
sending )
```

```
        * gsTCSGetTLClipInfo    0                                                   1
        2               0                               - Last get state
        *       Rtn                                     0               0
        150             0               1               0               file2   - 5 sec
A1 from file2
        *       ( Use last gsTCSGetTLNextState for this call )
        * gsTCSGetTLNextState 0                                                    1
        2               0                               - Use last state to get next
        *       Rtn                                     0
                1               3               0                               -
Channels used so far
        *       ( Copy prev gsTCSGetTLNextState into gsTCSGetTLClipInfo before
sending )
        * gsTCSGetTLClipInfo    0                                                   1
        3               0                               - Last get state
        *       Rtn                                     150             150
        210             0               1               0               file3   - 2 sec
A1 from file3
        *       ( Use last gsTCSGetTLNextState for this call )
        * gsTCSGetTLNextState 0                                                    1
        3               0                               - Use last state to get next
        *       Rtn                                     150
                0               1               0                               -
Channels used so far
        * Take the #MEDIACMD struct returned from gsTCSGetTLClipState and find the
next active
        * clip.  For the first clip in time line, send all zeroes.  Other then the first call,
        * all calls should include the position/channel bits from the previous
gsTCSGetTLNextState
        * call and (other then first call) gsTCSGetTLNextState should be call
immeadiately before
        * gsTCSGetTLClipInfo.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::arbID - Clip ID
        * <BR>  for MCMD2 -out- MEDIACMD::arbID - Next 8 character id and unc file
path seperated by NULL or 8 NULLs if clip list complete
        * <BR>  MEDIACMD::cfFlags - Set cfUsePosition|cfUseStart|cfUseEnd to search
next clip, set cfUsePosition & MEDIACMD::dwPosition for info at specified position
        * <BR>  MEDIACMD::dwPosition - Reference time code for time line
        * <BR>  MEDIACMD::dwStart - First frame of clip
        * <BR>  MEDIACMD::dwEnd - Last frame of clip
        * <BR>MEDIACMD::dwVideoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <BR>MEDIACMD::dwAudioChannels - Channels this clip exists in for the
dwStart/dwEnd range
```

```
        * <BR>MEDIACMD::dwInfoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <HR>
        */
        gsTCSGetTLClipInfo,
        /**
        * Get or change the information on a clip (currently for clip space only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  - not supported
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::arbID - Last returned clip id or 8 NULLs for first clip
        * <BR>  for MCMD2 -out- MEDIACMD::arbID - Next 8 character id and unc file
path seperated by NULL or 8 NULLs if clip list complete
        * <BR>  MEDIACMD::dwPosition - Starting timecode if known, else First frame of
clip
        * <BR>  MEDIACMD::dwStart - First frame of clip
        * <BR>  MEDIACMD::dwEnd - Last frame of clip
        * <BR>MEDIACMD::dwVideoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <BR>MEDIACMD::dwAudioChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <BR>MEDIACMD::dwInfoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <HR>
        */
        gsClipInfo,                             // Same as above for named clip or
current (arbID - name, dwStart, dwEnd if avail)
        /**
        * Create a virtual copy of a clip from a current clip.  Must change at least name
        * to succeed.  To affect the source clip, use #gsClipInfo
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  Requires MEDIACMD::cfFlags set to affect stored clip info for each
member
        * <BR>  MEDIACMD::arbID - Source ClipID[8 bytes], NULL, New ClipID[8 bytes]
- min size 17 bytes.
        * <BR>  MEDIACMD::dwStart - First frame of new clip (referenced from source
clip)
        * <BR>  MEDIACMD::dwEnd - Last frame of new clip (referenced from source
clip)
        * <BR>MEDIACMD::dwVideoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <BR>MEDIACMD::dwAudioChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * <BR>MEDIACMD::dwInfoChannels - Channels this clip exists in for the
dwStart/dwEnd range
        * \li cmdType::ctGetValue
        * <BR>- no supported
```

```
        * <HR>
        */
        gsClipCopy,                             // Copy current clip to specified name


        /**
        * Returns the available audio channels (read only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- no supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Available channels
        * <HR>
        */
        gsAudChan = 100,                // Available audio channels (dwPosition)
        /**
        * Returns the available video channels (read only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- no supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Available channels
        * <HR>
        */
        gsVidChan,                              // Available video channels (dwPosition)
        /**
        * Returns the available information channels (read only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- no supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Available channels
        * <HR>
        */
        gsInfChan,                              // Available info channels (dwPosition)
        /**
        * Return or set the selected audio channels
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected channels
        * <BR>MEDIACMD::dwStart - Available channels for selection
        * <HR>
        */
        gsAudSelect,                    // Selected audio channels (dwPosition,
supported dwStart)
        /**
        * Return or set the selected video channels
```

```
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected channels
        * <BR>MEDIACMD::dwStart - Available channels for selection
        * <HR>
        */
        gsVidSelect,                    // Selected video channels (dwPosition,
supported dwStart)
        /**
        * Return or set the selected information channels
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected channels
        * <BR>MEDIACMD::dwStart - Available channels for selection
        * <HR>
        */
        gsInfSelect,                    // Selected info channels (dwPosition, supported
dwStart)
        /**
        * Return or set the audio channels for the next edit
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel edit selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected edit channels
        * <BR>MEDIACMD::dwStart - Available channels for edit
        * <HR>
        */
        gsAudEdit,                          // Edit ready audio channels (dwPosition,
supported dwStart)
        /**
        * Return or set the video channels for the next edit
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel edit selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected edit channels
        * <BR>MEDIACMD::dwStart - Available channels for edit
        * <HR>
        */
        gsVidEdit,                          // Edit ready video channels (dwPosition,
supported dwStart)
        /**
        * Return or set the information channels for the next edit
        * <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel edit selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected edit channels
        * <BR>MEDIACMD::dwStart - Available channels for edit
        * <HR>
        */
        gsInfEdit,                          // Edit ready info channels (dwPosition,
supported dwStart)
        /**
        * Return or set the information channels for the next edit
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New channel edit selection
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently selected edit channels
        * <BR>MEDIACMD::dwStart - Available channels for edit
        * <HR>
        */
        gsEditMode,                         // Edit to use assemble or insert

        // MetaData set/retieve
        /**
        * Access one meta data element for the current media. <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwEnd - GS_TRUE if the element exists,
GS_NOT_SUPPORTED if not
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwEnd - GS_TRUE if the element exists,
GS_NOT_SUPPORTED if not
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaData = 150,
        /**
```

```
        * Access one meta data element for the directory of the current media
(./Default.xml). <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaDataDirectory,
        /**
        * Access one meta data element for the drive/volume of the current media
(/Default.xml). <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaDataVolume,
        /**
        * Access one meta data element for the defult metadata of the current user
        * (HKEY_CURRENT_USER windows, /home/user/default.xml unix). <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
```

```
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaDataCurrentUser,
        /**
        * Access one meta data element for the defult metadata of the current user
        * (HKEY_LOCAL_MACHINE windows, /var/metadata/default.xml unix). <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaDataLocalMachine,
        /**
        * Access one meta data element for the defult metadata for the facility
        * (Requires group or facility media proxy and database). <BR>
        * See the enum #vvwInfoMetaTypes in vwwTypes.h
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
```

```
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - ID = #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::dwStart - value for #vvwInfoMetaTypes::vvwiTimeCode ..
#vvwInfoMetaTypes::vvwiGamma1000
        * <BR> MEDIACMD::arbID - value for #vvwInfoMetaTypes::vvwiFileName ..
#vvwInfoMetaTypes::vvwiFrameAttribute
        * <HR>
        */
        gsMetaDataGlobal,
        /**
        * Write or read the current metadata structure from the XML on the disk
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - 0 = write file
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - 0 = read file
        * <HR>
        */
        gsMetaDataReadWrite,

        // Audio settings us dwAudioChannels (except ltc)
        /**
        * Audio input select
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New audio input
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
        * #GS_AUDSELECT_BALANCED_10      #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
        * #GS_AUDSELECT_EMBEDDED
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current input
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
        * #GS_AUDSELECT_BALANCED_10      #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
        * #GS_AUDSELECT_EMBEDDED
        * <BR>MEDIACMD::dwStart - Bit array of available inputs, see above or
#GS_AUDSELECT_NONE
        * <BR>MEDIACMD::dwAudioChannels - Channels requested
        * <HR>
        */
        gsAudInSelect = 200,   // Audio Input Select (dwPosition, available = dwStart)
        /**
        * Audio output select (in general all outputs are active)
        * <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New audio output
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
        * #GS_AUDSELECT_BALANCED_10      #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
        * #GS_AUDSELECT_EMBEDDED
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current output
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
        * #GS_AUDSELECT_BALANCED_10      #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
        * #GS_AUDSELECT_EMBEDDED
        * <BR>MEDIACMD::dwStart - Bit array of available outputs, see above or
#GS_AUDSELECT_NONE
        * <BR>MEDIACMD::dwAudioChannels - Channels requested
        * <HR>
        */
        gsAudOutSelect,                         // Audio Output Select (dwPosition,
available = dwStart)

        /**
        * Audio input level (gain)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels affected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwStart - Minimum level (usually 0)
        * <BR>MEDIACMD::dwEnd - Maximum level (usually 65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * <HR>
        */
        gsAudInputLevel,                // Input level setting (16 bit) (dwPosition, min =
dwStart, max = dwEnd)
        /**
        * Audio output level (master)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels affected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwStart - Minimum level (usually 0)
        * <BR>MEDIACMD::dwEnd - Maximum level (usually 65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * <HR>
        */
```

```
        gsAudOutputLevel,               // Output level setting (16 bit) (dwPosition, min
= dwStart, max = dwEnd)
        /**
        * Audio advance level (advanced cue head master) - Not Supported
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels affected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Level (0-65535)
        * <BR>MEDIACMD::dwStart - Minimum level (usually 0)
        * <BR>MEDIACMD::dwEnd - Maximum level (usually 65535)
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * <HR>
        */
        gsAudAdvanceLevel,              // ??? (Not Supported) (dwPosition, min =
dwStart, max = dwEnd)
        /**
        * Audio output phase
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Phase offset (default = 0) (0-65520 = degrees
* 182)
        * <BR>MEDIACMD::dwAudioChannels - Channels affected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Phase offset (0-65520 = degrees * 182)
        * <BR>MEDIACMD::dwStart - Minimum phase available (usually 0)
        * <BR>MEDIACMD::dwEnd - Maximum phase available (usually 65520 = 360 *
182)
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * <HR>
        */
        gsAudOutPhase,                          // In samples (dwPosition, min =
dwStart, max = dwEnd)
        /**
        * Audio advance phase (advanced cue head master) - Not Supported
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Phase offset (default = 0) (0-65520 = degrees
* 182)
        * <BR>MEDIACMD::dwAudioChannels - Channels affected
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Phase offset (0-65520 = degrees * 182)
        * <BR>MEDIACMD::dwStart - Minimum phase available (usually 0)
        * <BR>MEDIACMD::dwEnd - Maximum phase available (usually 65520 = 360 *
182)
        * <BR>MEDIACMD::dwAudioChannels - Channels effected
        * <HR>
        */
```

```
        gsAudOutAdvancePhase,          // ??? (Not Supported) in samples (dwPosition,
min = dwStart, max = dwEnd)
        /**
        * Audio crossfade time (clip effect overlap) - Not Supported
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Length of crossfade in milliseconds
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Length of crossfade in milliseconds
        * <BR>MEDIACMD::dwStart - Minimum crossfade length (usually 0 = cut)
        * <BR>MEDIACMD::dwEnd - Maximum crossfade length (depends on device)
        * <HR>
        */
        gsAudCrossFadeTime,            // Audio cross fade duration (dwPosition, min =
dwStart, max = dwEnd)
        /**
        * Enable Ltc on an audio channel
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - #GS_ENABLE or #GS_DISABLE
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for LTC
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Is LTC enabled
        * <HR>
        */
        gsAudLtcEnable,                        // Ltc enabled (dwposition)
        /**
        * Set audio channel to use for LTC input if enabled.  Currently will set
        * LTC output to same channel on all VVW drivers.
        *
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for LTC
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit channel is using for LTC
        * <HR>
        */
        gsAudInLtcChannel,             // Ltc channel, -1 if disabled (dwPosition,
available = dwStart)
        /**
        * Set audio channel to use for LTC output if enabled.  Currently will set
        * LTC input to same channel on all VVW drivers.
        *
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for LTC
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit channel is using for LTC
        * <HR>
```

```
        */
        gsAudOutLtcChannel,             // Ltc channel, -1 if disabled (dwPosition,
available = dwStart)
        /**
        * Enable Dtmf on an audio channel
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - #GS_ENABLE or #GS_DISABLE
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for DTMF
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Is DTMF enabled
        * <HR>
        */
        gsAudDtmfEnable,                // Dtmf enabled (dwposition)
        /**
        * Set audio channel to use for DTMF input if enabled.  Currently will set
        * DTMF output to same channel on all VVW drivers.
        *
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for DTMF
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit channel is using for DTMF
        * <HR>
        */
        gsAudInDtmfChannel,             // Dtmf channel, -1 if disabled (dwPosition,
available = dwStart)
        /**
        * Set audio channel to use for DTMF output if enabled.  Currently will set
        * DTMF input to same channel on all VVW drivers.
        *
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit for channel to use for DTMF
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwAudioChannels - Bit channel is using for DTMF
        * <HR>
        */
        gsAudOutDtmfChannel, // Dtmf channel, -1 if disabled (dwPosition, available =
dwStart)
        /**
        * Return the last known RMS and peak value of the audio output.  Max 2
channels returned
        * per call.  2 channels should always be requested
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- Not Supported
        * \li cmdType::ctGetValue
        * <BR>-in- MEDIACMD::dwAudioChannels - Requested channels to check
```

```
        * <BR>MEDIACMD::dwStart - HIWORD=RMS channel +1, LOWORD=RMS
channel +0 (range 0-65535)
        * <BR>MEDIACMD::dwEnd - HIWORD=Peak channel +1, LOWORD=Peak
channel +0 (range 0-65535)
        * <BR>MEDIACMD::dwPosition - duplicates #MEDIACMD::dwStart
        * <HR>
        */
        gsAudWavePeakRMS,               // Current play or in peak|rms 0-
(dwStart:HiWord|LoWord), 1-(dwEnd:HiWord|LoWord)
        /**
        * Get / Set the current bit rate for recording audio
        * per call
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- Not Supported
        * \li cmdType::ctGetValue
        * <BR>-in- MEDIACMD::dwAudioChannels - Requested channels to check
        */
        gsAudInputBitRate,                      //Sets the bit rate for audio records
(Argus)
        /**
        * Set the audio input sample rate
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New sample rate (typically 48000 or 96000)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current sample rate (use
cmdType::ctValueSupported to get list)
        * <BR>MEDIACMD::dwStart - Lowest supported sample rate
        * <BR>MEDIACMD::dwEnd - Highest supported sample rate
        * <HR>
        */
        gsAudInputSampleRate,
        /**
        * Audio input mode
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - See sony defines
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - See sony defines
        * <HR>
        */
        gsAudInputMode,
        /**
        * Audio input head room
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Set required headroom
        * \li cmdType::ctGetValue
```

```
* <BR>MEDIACMD::dwPosition - Current headroom
* <BR>MEDIACMD::dwStart - Min headroom
* <BR>MEDIACMD::dwEnd - Max Headroom ;)
* <HR>
*/
gsAudInputHeadRoom,
/**
* Audio input original - see sony def
* <BR>
* \li cmdType::ctSetValue
* <BR>MEDIACMD::dwPosition -
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition -
* <HR>
*/
gsAudInputOriginal,
/**
* Enable and disable audio input error protection
* <BR>
* \li cmdType::ctSetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE protection enabled, else GS_FALSE
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE protection enabled, else GS_FALSE
* <HR>
*/
gsAudInputErrorProtect,
/**
* Does the audio input bit stream contain a copyright flag
* <BR>
* \li cmdType::ctSetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE copyright flag is set, else GS_FALSE
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE copyright flag is set, else GS_FALSE
* <HR>
*/
gsAudInputCopyright,
/**
* Audio input is in slave mode
* <BR>
* \li cmdType::ctSetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE if in slave mode, else GS_FALSE
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition - GS_TRUE if in slave mode, else GS_FALSE
* <HR>
*/
gsAudInputSlave,
/**
* Audio bass setting, hardware dependant
* <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0..65536, 32768 being nominal and -1 being
default
        * <BR>MEDIACMD::dwAudioChannels - Bit(s) in use
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 0..65536, 32768 being nominal and -1 being
not supported
        * <BR>MEDIACMD::dwAudioChannels - Bit(s) in use
        * <HR>
        */
        gsAudInputBass,
        /**
        * Audio treble setting, hardware dependant
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0..65536, 32768 being nominal and -1 being
default
        * <BR>MEDIACMD::dwAudioChannels - Bit(s) in use
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 0..65536, 32768 being nominal and -1 being
not supported
        * <BR>MEDIACMD::dwAudioChannels - Bit(s) in use
        * <HR>
        */
        gsAudInputTreble,
        /**
        * What audio channels are available, selected and valid
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Available
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Audio channel bits with valid inputs
        * <BR>MEDIACMD::dwStart - Audio channel bits on primary audio selection
        * <BR>MEDIACMD::dwEnd - Audio channel bits being monitored
        * <BR>MEDIACMD::dwAudioChannels - Audio channel bits available
        * <HR>
        */
        gsAudInputStatus,



        /**
        * Freeze the video output
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Freeze type #GS_VIDFREEZE_NOT_FROZEN,
#GS_VIDFREEZE_FIELD0, #GS_VIDFREEZE_FIELD1, #GS_VIDFREEZE_FRAME
        * <BR>MEDIACMD::dwVideoChannels - Bit(s) for the channel(s) to freeze
        * \li cmdType::ctGetValue
```

```
        * <BR>MEDIACMD::dwAudioChannels - Bit(s) for the channel(s) currently frozen
        * <HR>
        */
        gsVidFreeze = 300,              // Freeze video 0-un, 1 field, 2 field, 3 both
(dwPosition)
        /**
        * Set DDR into pre read (read before write) mode - requires 2 or more channels
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Channel to use as output
        * <BR>MEDIACMD::dwStart - #GS_ENABLE or #GS_DISABLE
        * <BR>MEDIACMD::dwVideoChannels - Channels to record
        * <BR>MEDIACMD::dwAudioChannels - Channels to record
        * <BR>MEDIACMD::dwInfoChannels - Channels to record
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwStart - #GS_ENABLE or #GS_DISABLE - if not enabled,
the rest does not matter
        * <BR>MEDIACMD::dwPosition - Channel in use as output
        * <BR>MEDIACMD::dwVideoChannels - Channels recording
        * <BR>MEDIACMD::dwAudioChannels - Channels recording
        * <BR>MEDIACMD::dwInfoChannels - Channels recording
        * <HR>
        */
        gsVidPreReadMode,               // Setup for preread mode (dwPosition =
channel, dwStart = used channels, dwEnd = available channels);
        /**
        * First field recorded in edit
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Edit start field (#GS_FIELD2 for second, else
first)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Edit start field (#GS_FIELD2 for second, else
first)
        * <HR>
        */
        gsVidEditField,                 // Starting field for an edit (2=2nd, else 1st)
(dwPosition = channel, dwStart = available channels)
        /**
        * Record frames or fields
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_FIELD record single field, else record
frames (default - frames (both fields))
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_FIELD recording single field, else
recording frames (default - frames (both fields))
        * <HR>
        */
```

```
        gsVidRecFrame,                  // Record frame or field (dwPosition = channel,
dwStart = available channels)
        /**
        * Play frames or fields
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_FIELD play single field, else play frames
(default - frames (both fields))
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_FIELD recording fields, else recording
frames (default - frames (both fields))
        * <HR>
        */
        gsVidPlayFrame,                         // Play frame or field (dwPosition =
channel, dwStart = available channels)
        /**
        * Disable video edit to edit passthrough
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE Always in playback mode, else if
#GS_FALSE then will passthough video
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE Always in playback mode, else if
#GS_FALSE then will passthough video
        * <HR>
        */
        gsVidNoEE,                              // No E to E mode allowed (dwPosition =
channel, dwStart = available channels)
        /**
        * Enable superimposed tc/state/menu output in video
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE Superimpose, else normal video
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE Superimpose, #GS_TRUE Normal
Video, #GS_NOT_SUPPORTED cannot superimpose
        * <HR>
        */
        gsVidSuperimpose,               // Super tc/state data on monitor output
(dwPosition = channel, dwStart = available channels)

        /**
        * Set the type of up down convert to do
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_ANALOGMONITORMETHOD_DIRECT,
#GS_ANALOGMONITORMETHOD_SD,
        *   #GS_ANALOGMONITORMETHOD_HD720,
#GS_ANALOGMONITORMETHOD_HD1080, #GS_ANALOGMONITORMETHOD_FLIP720,
```

```
        *   #GS_ANALOGMONITORMETHOD_FLIP1080
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_ANALOGMONITORMETHOD_DIRECT,
#GS_ANALOGMONITORMETHOD_SD,
        *   #GS_ANALOGMONITORMETHOD_HD720,
#GS_ANALOGMONITORMETHOD_HD1080, #GS_ANALOGMONITORMETHOD_FLIP720,
        *   #GS_ANALOGMONITORMETHOD_FLIP1080 or GS_NOT_SUPPORTED
        * <HR>
        */
        gsVidAnalogMonitorSDType,
        /**
        * Select the output type of the analog SD (Composite, SMPTE, RGB)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition -
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition -
        * <HR>
        */
        gsVidAnalogMonitorHDType,
        /**
        * Select the output type of the analog HD (RGB, SMPTE, xVidRGB)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition -
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition -
        * <HR>
        */
        gsVidAnalogMonitorMethod,
        /**
        * Select the method for up converting to HD
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_UPCONVERT_ANAMORPHIC,
#GS_UPCONVERT_PILLARBOX,
        *    #GS_UPCONVERT_ZOOM14x9, #GS_UPCONVERT_LETTERBOX,
#GS_UPCONVERT_ZOOMWIDE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_UPCONVERT_ANAMORPHIC,
#GS_UPCONVERT_PILLARBOX,
        *    #GS_UPCONVERT_ZOOM14x9, #GS_UPCONVERT_LETTERBOX,
#GS_UPCONVERT_ZOOMWIDE
        * <HR>
        */
        gsVidAnalogMonitorUpMode,
        /**
        * Select the method for down converting to SD
        * <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_DOWNCONVERT_LETTERBOX,
#GS_DOWNCONVERT_CROP, #GS_DOWNCONVERT_ANAMORPHIC
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_DOWNCONVERT_LETTERBOX,
#GS_DOWNCONVERT_CROP, #GS_DOWNCONVERT_ANAMORPHIC
        * <HR>
        */
        gsVidAnalogMonitorDownMode,

        /**
        * Set/Get the current pan scan pos and zoom
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - when to do it (0xFFFFFFFF == immeadiate)
        * <BR>MEDIACMD::dwStart - X (upper bit 0x80000000 is FLIP)
        * <BR>MEDIACMD::dwEnd - Y (upper bit 0x80000000 is FLIP)
        * <BR>MEDIACMD::dwSpeed - Z
        * <BR>MEDIACMD::dwAudioChannels -  X Aspect
        * <BR>MEDIACMD::dwVideoChannels -  Y Aspect
        * <BR>MEDIACMD::dwInfoChannels - Rotate
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - when to do it
        * <BR>MEDIACMD::dwStart - X (upper bit 0x80000000 is FLIP)
        * <BR>MEDIACMD::dwEnd - Y (upper bit 0x80000000 is FLIP)
        * <BR>MEDIACMD::dwSpeed - Z
        * <BR>MEDIACMD::dwAudioChannels -  X Aspect
        * <BR>MEDIACMD::dwVideoChannels -  Y Aspect
        * <BR>MEDIACMD::dwInfoChannels - Rotate
        * <HR>
        */
        gsVidPanScanZoom,

        // Video settings use dwVideoChannels
        /**
        * Select video input
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input to use
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
        * #GS_VIDSELECT_COMPONENT_YUV,
#GS_VIDSELECT_COMPONENT_YUV_M2, #GS_VIDSELECT_COMPONENT_YUV_SMPTE,
#GS_VIDSELECT_COMPONENT_RGB,
        * #GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL,
#GS_VIDSELECT_SDTI, #GS_VIDSELECT_NONE
        * \li cmdType::ctGetValue
```

```
        * <BR>MEDIACMD::dwPosition - Video input to use
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
        * #GS_VIDSELECT_COMPONENT_YUV,
#GS_VIDSELECT_COMPONENT_YUV_M2, #GS_VIDSELECT_COMPONENT_YUV_SMPTE,
#GS_VIDSELECT_COMPONENT_RGB,
        * #GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL,
#GS_VIDSELECT_SDTI, #GS_VIDSELECT_NONE
        * <BR>MEDIACMD::dwStart - Supported video inputs (bit array using defines
from dwPosition)
        * <HR>
        */
        gsVidInSelect = 400,    // Select video input source (dwPosition, supported =
dwStart)
        /**
        * Select video input genlock type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input lock type use
#GS_VIDLOCKTYPE_VTR or #GS_VIDLOCKTYPE_BROADCAST
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input lock type use
#GS_VIDLOCKTYPE_VTR or #GS_VIDLOCKTYPE_BROADCAST
        * <BR>MEDIACMD::dwStart - Supported video inputs (bit array using defines
from dwPosition)
        * <HR>
        */
        gsVidInLockType,                // Input channel lock type (1-Broadcast, 0-VTR)
        /**
        * Input TBC - Setup (~brightness) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Setup
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Setup
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidInSetup,                   // Input channel Setup (16 bit unsigned)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Input TBC - Video (~contrast) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Video
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Video
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
```

```
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidInVideo,                   // Input channel Video (16 bit unsigned)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Input TBC - Hue (~color angle) degrees * 182.  Normal range: 0-65520
(0x0000-0xfff0)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Hue
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Hue
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65520)
        * <HR>
        */
        gsVidInHue,                     // Input channel Hue (16 bit unsigned)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Input TBC - Chroma (~saturation) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Chroma
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidInChroma,                  // Input channel Chroma (16 bit unsigned)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Input TBC - U Chroma or Cb or Y'CrCb Normal range: 0-65535 (0x0000-0xffff)
        * Normally only effects the component video or D1 Serial inputs.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC U (Cb) Chroma
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC U (Cb) Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidInUChroma,                 // Input channel U Component Chroma
(16 bit unsigned) (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Input TBC - V Chroma or Cr or Y'CrCb Normal range: 0-65535 (0x0000-0xffff)
        * Normally only effects the component video or D1 Serial inputs.
```

```
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC V (Cr) Chroma
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video input TBC V (Cr) Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidInVChroma,                         // Input channel V Component Chroma
(16 bit unsigned) (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Remove color from input signal (black and white luminance data only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal will have no chroma, if 0, normal
signal
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal will have no chroma, if 0, normal
signal
        * <HR>
        */
        gsVidInColorKiller,                 // Kill colour on input (dwPosition bool)
        /**
        * Automatic gain control
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal adjust gain automatically, if 0, will
us cmdGetSetValue::gsVidInSetup and cmdGetSetValue::gsVidInVideo
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal adjust gain automatically, if 0, will
us cmdGetSetValue::gsVidInSetup and cmdGetSetValue::gsVidInVideo
        * <HR>
        */
        gsVidInAGC,                             // Input channel automatic gain control
(dwPosition bool)
        /**
        * Maximum input bandwidth setting
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBAND_STANDARD,
#GS_VIDBAND_MEDIUM, #GS_VIDBAND_HIGH, #GS_VIDBAND_NOTCH
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBAND_STANDARD,
#GS_VIDBAND_MEDIUM, #GS_VIDBAND_HIGH, #GS_VIDBAND_NOTCH
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above
        * <HR>
        */
```

```
        gsVidInBandwidth,               // Signal bandwidth (dwPosition, supported =
dwStart)
        /**
        * Black type (NTSC only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBLACK_SETUP,
#GS_VIDBLACK_CRYSTAL, #GS_VIDBLACK_SUPER
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBLACK_SETUP,
#GS_VIDBLACK_CRYSTAL, #GS_VIDBLACK_SUPER
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above
        * <HR>
        */
        gsVidInBlack,                   // Black level (dwPosition, supported = dwStart)
        /**
        * White type (NTSC only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDWHITE_CLAMP,
#GS_VIDWHITE_SCALE, #GS_VIDWHITE_FREE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDWHITE_CLAMP,
#GS_VIDWHITE_SCALE, #GS_VIDWHITE_FREE
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above
        * <HR>
        */
        gsVidInWhite,                   // Max white (dwPosition, supported = dwStart)
        /**
        * Input digital signal coring.  Removal of low order bits to remove DAC aliasing
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Remove bottom 0, 1 or 2 bits of digitized signal
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Remove bottom 0, 1 or 2 bits of digitized signal
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above (0 always supported)
        * <HR>
        */
        gsVidInCoring,                  // Input channel coring 0, 1 or 2 bits
(dwPosition, supported = dwStart)
        /**
        * Remove (smooth) 100% signal spikes
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0 leave signal intact, 1 smooth
        * \li cmdType::ctGetValue
```

```
        * <BR>MEDIACMD::dwPosition - 0 leave signal intact, 1 smooth
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above (0 always supported)
        * <HR>
        */
        gsVidInPeaking,                 // (dwPosition, supported = dwStart)
        /**
        * Set video transition sharpness
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0-7, 0-100, 0-65535)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Video digitizing sharpness
        * <BR>MEDIACMD::dwStart - Lowest possible sharpness
        * <BR>MEDIACMD::dwEnd - Highest possible sharpness
        * <HR>
        */
        gsVidInSharpness,               // (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Set video gamma curve
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically -32768->+32768)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Gamma curve weighting or offset
        * <BR>MEDIACMD::dwStart - Lowest possible sharpness
        * <BR>MEDIACMD::dwEnd - Highest possible sharpness
        * <HR>
        */
        gsVidInGamma,                   // (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Video input signal format.  May be incorrect depending on some
        * hardware setups.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- Not supported, please use #gsSignalFormat to set channel format to
        * match input
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_SIGFORM_NTSC #GS_SIGFORM_PAL
        * #GS_SIGFORM_CCIR_NTSC #GS_SIGFORM_CCIR_PAL
        * #GS_SIGFORM_1035i_30_260M #GS_SIGFORM_1035i_30X_260M
        * #GS_SIGFORM_1080i_30 #GS_SIGFORM_1080i_30X #GS_SIGFORM_1080i_25
#GS_SIGFORM_1080i_24 #GS_SIGFORM_1080i_24X
        * #GS_SIGFORM_1080_30 #GS_SIGFORM_1080_30X #GS_SIGFORM_1080_25
#GS_SIGFORM_1080_24 #GS_SIGFORM_1080_24X
        * #GS_SIGFORM_720_60 #GS_SIGFORM_720_60X
#GS_SIGFORM_NOT_PRESENT
```

```
        * <HR>
        */
        gsVidInSignalFormat,    // Input signal format (-1 for not present or bad setup)

        /**
        * Main TBC - Setup (~brightness) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC Setup
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TBC Setup
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidSetup = 500,               // Video 'Setup' (16 bit signed) (dwPosition,
min=dwStart, max=dwEnd)
        /**
        * Main TBC - Video (~contrast) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC Video
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TBC Video
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidVideo,                             // Video 'Video' (16 bit signed)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Main TBC - Hue (~color angle) degrees * 182.  Normal range: 0-65520
(0x0000-0xfff0)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC Hue
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Hue
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65520)
        * <HR>
        */
        gsVidHue,                               // Video 'Hue' (16 bit signed)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Main TBC - Chroma (~saturation) Normal range: 0-65535 (0x0000-0xffff)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC Chroma
```

```
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TBC Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidChroma,                    // Video 'Chroma' (16 bit signed) (dwPosition,
min=dwStart, max=dwEnd)
        /**
        * Main TBC - U Chroma or Cb or Y'CrCb Normal range: 0-65535 (0x0000-0xffff)
        * Normally only effects the component video or D1 Serial paths.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC U (Cb) Chroma
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TBC U (Cb) Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidUChroma,                   // Input channel U Component Chroma (16 bit
signed) (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Main TBC - V Chroma or Cr or Y'CrCb Normal range: 0-65535 (0x0000-0xffff)
        * Normally only effects the component video or D1 Serial paths.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TBC V (Cr) Chroma
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TBC V (Cr) Chroma
        * <BR>MEDIACMD::dwStart - Lowest possible value (usually 0)
        * <BR>MEDIACMD::dwEnd - Highest possible value (usually 65535)
        * <HR>
        */
        gsVidVChroma,                   // Input channel V Component Chroma (16 bit
signed) (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Maximum channel bandwidth setting
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBAND_STANDARD,
#GS_VIDBAND_MEDIUM, #GS_VIDBAND_HIGH, #GS_VIDBAND_NOTCH
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBAND_STANDARD,
#GS_VIDBAND_MEDIUM, #GS_VIDBAND_HIGH, #GS_VIDBAND_NOTCH
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above
        * <HR>
        */
```

```
        gsVidBandwidth,                         // Signal bandwidth SEE
gsVidInBandwidth (dwPosition, supported=dwStart)
        /**
        * Black type (NTSC only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBLACK_SETUP,
#GS_VIDBLACK_CRYSTAL, #GS_VIDBLACK_SUPER
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses #GS_VIDBLACK_SETUP,
#GS_VIDBLACK_CRYSTAL, #GS_VIDBLACK_SUPER
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above
        * <HR>
        */
        gsVidBlackSetup,                // Super black, Crystal Black, NTSC Setup SEE
gsVidInBlack (dwPosition, supported=dwStart)
        /**
        * Remove color from signal path (black and white luminance data only)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal will have no chroma, if 0, normal
signal
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - If 1, signal will have no chroma, if 0, normal
signal
        * <HR>
        */
        gsVidColor,                             // Color signal or black and white
(dwPosition)

        /**
        * Select video output
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Video output to use
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
        * #GS_VIDSELECT_COMPONENT_YUV,
#GS_VIDSELECT_COMPONENT_YUV_M2, #GS_VIDSELECT_COMPONENT_YUV_SMPTE,
#GS_VIDSELECT_COMPONENT_RGB,
        * #GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL,
#GS_VIDSELECT_SDTI, #GS_VIDSELECT_NONE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current video output
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
```

```
        * #GS_VIDSELECT_COMPONENT_YUV,
#GS_VIDSELECT_COMPONENT_YUV_M2, #GS_VIDSELECT_COMPONENT_YUV_SMPTE,
#GS_VIDSELECT_COMPONENT_RGB,
        * 7#GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL,
#GS_VIDSELECT_SDTI, #GS_VIDSELECT_NONE
        * <BR>MEDIACMD::dwStart - Supported video inputs (bit array using defines
from dwPosition)
        * <HR>
        */
        gsVidOutSelect = 600,  // Select main out (See gsVidInSelect)
        /**
        * Enable genlock (video black timing signal)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 1 using external ref genlock, 0 free running on
internal clock
        * (see gsGetSetCmdValue::gsVidOutGenlockSource)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 1 using external ref genlock, 0 free running on
internal clock
        * (see gsGetSetCmdValue::gsVidOutGenlockSource)
        * <BR>MEDIACMD::dwStart - If 1, external genlock supported
        * <HR>
        */
        gsVidOutGenlock,                // Genlock enable (dwPosition,
dwStart=supported sources)
        /**
        * Select genlock (video black timing signal) source
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Genlock source to use #GS_LOCKSRC_NONE,
#GS_LOCKSRC_EXTIN,
        * #GS_LOCKSRC_INPUT, #GS_LOCKSRC_CVBS (composite video),
#GS_LOCKSRC_SVIDEO (svhs),
        * #GS_LOCKSRC_IN_Y (y of component in), #GS_LOCKSRC_SDI (D1 Digital In)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Genlock source to use #GS_LOCKSRC_NONE,
#GS_LOCKSRC_EXTIN,
        * #GS_LOCKSRC_INPUT, #GS_LOCKSRC_CVBS (composite video),
#GS_LOCKSRC_SVIDEO (svhs),
        * #GS_LOCKSRC_IN_Y (y of component in), #GS_LOCKSRC_SDI (D1 Digital In)
        * <BR>MEDIACMD::dwStart - Supported genlock inputs (bit array using defines
from dwPosition)
        * <HR>
        */
        gsVidOutGenlockSource,
        /**
        * Select genlock type (quality)
        * <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Genlock lock type use
#GS_VIDLOCKTYPE_VTR or #GS_VIDLOCKTYPE_BROADCAST
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Genlock lock type use
#GS_VIDLOCKTYPE_VTR or #GS_VIDLOCKTYPE_BROADCAST
        * <BR>MEDIACMD::dwStart - Supported video inputs (bit array using defines
from dwPosition)
        * <HR>
        */
        gsVidOutLockType,               // Genlock lock type (0-Broadcast, 1-VTR)
(dwPosition )
        /**
        * Horizontal output phase
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0->65535 or -32768->32768)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Horizontal phase setting
        * <BR>MEDIACMD::dwStart - Lowest possible horizontal phase
        * <BR>MEDIACMD::dwEnd - Highest possible horizontal phase
        * <HR>
        */
        gsVidOutHPhase,                         // Horizontal Phase (16 bit signed)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Video genlock subcarrier phase timing
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0->65520 == degrees * 182)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Sub carrier phase setting
        * <BR>MEDIACMD::dwStart - Lowest possible sub carrier phase
        * <BR>MEDIACMD::dwEnd - Highest possible sub carrier phase
        * <HR>
        */
        gsVidOutSubCarrier,             // Sub Carrier Phase (16 bit signed) (dwPosition,
min=dwStart, max=dwEnd)
        /**
        * Digital output signal coring.  Removal of low order bits to remove DAC aliasing
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Remove bottom 0, 1 or 2 bits of digitized signal
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Remove bottom 0, 1 or 2 bits of digitized signal
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above (0 always supported)
```

```
        * <HR>
        */
        gsVidOutCoring,                         // Core 0, 1 or 2 bits from signal (0, 1, 2)
(dwPosition, supported=dwStart)
        /**
        * Remove (smooth) 100% signal spikes on output
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0 leave signal intact, 1 smooth
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 0 leave signal intact, 1 smooth
        * <BR>MEDIACMD::dwStart - Bit array of allowable values as defined for
dwPosition above (0 always supported)
        * <HR>
        */
        gsVidOutPeaking,                // Peaking (dwPosition, supported=dwStart)
        /**
        * Generic advanced adjustment 1 (hardware dependant)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0->65535 or -32768->32768)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Adjust 1 setting
        * <BR>MEDIACMD::dwStart - Lowest possible adjust 1 setting
        * <BR>MEDIACMD::dwEnd - Highest possible adjust 1 setting
        * <HR>
        */
        gsVidOutAdjust1,                //
        /**
        * Generic advanced adjustment 2 (hardware dependant)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0->65535 or -32768->32768)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Adjust 2 setting
        * <BR>MEDIACMD::dwStart - Lowest possible adjust 2 setting
        * <BR>MEDIACMD::dwEnd - Highest possible adjust 2 setting
        * <HR>
        */
        gsVidOutAdjust2,                //
        /**
        * Genlock output delay (not currently used)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Depends on cmdType::ctGetValue-
MEDIACMD::dwStart->MEDIACMD::dwEnd (typically 0->65535 or -32768->32768)
        * \li cmdType::ctGetValue
```

```
        * <BR>MEDIACMD::dwPosition - Genlock timing delay
        * <BR>MEDIACMD::dwStart - Lowest possible delay
        * <BR>MEDIACMD::dwEnd - Highest possible delay
        * <HR>
        */
        gsVidOutGenlockDelay, //
        /**
        * Video output genlock input signal format.  May be incorrect depending
        * on some hardware setups.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>- Not supported, please use #gsSignalFormat to set channel format to
        * match input
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_SIGFORM_NTSC #GS_SIGFORM_PAL
        * #GS_SIGFORM_CCIR_NTSC #GS_SIGFORM_CCIR_PAL
        * #GS_SIGFORM_1035i_30_260M #GS_SIGFORM_1035i_30X_260M
        * #GS_SIGFORM_1080i_30 #GS_SIGFORM_1080i_30X #GS_SIGFORM_1080i_25
#GS_SIGFORM_1080i_24 #GS_SIGFORM_1080i_24X
        * #GS_SIGFORM_1080_30 #GS_SIGFORM_1080_30X #GS_SIGFORM_1080_25
#GS_SIGFORM_1080_24 #GS_SIGFORM_1080_24X
        * #GS_SIGFORM_720_60 #GS_SIGFORM_720_60X
#GS_SIGFORM_NOT_PRESENT
        * <HR>
        */
        gsVidOutLockSignalFormat,       // Genlock input signal format (-1 for not present
or bad setup)
        /**
        * When video input is in DualLink, the out switchs to dual link to.  If this is set,
then
        * the output will stay in single link and convert the dual link 4:4:4 to 4;2:2
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE, #GS_FALSE
        * match input
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE, #GS_FALSE
        * <HR>
        */
        gsVidOutDisableDualLink,        // Disable dual link out when in dual link in
        /**
        * Set the output of the Kona to show a wipe or dissolve against the current
frame
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0=dissolve, 1=Wipe
        * <BR>MEDIACMD::dwStart - Wipe Type, 0=horiz,1=vert,2=upperright,
3=upperleft
```

```
        * <BR>MEDIACMD::dwEnd - Wipe amount, 0..65535 (0..100%) where it is
percent of stored frame (e.g. 0=showinput,65535=showframe)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 0=dissolve, 1=Wipe
        * <HR>
        */
        gsVidOutReferenceWipeMix,               // Wipe or Mix between the input and
the frame on disk



        // Compression and internal signal parameters
        /**
        *       Size of picture   Y (Vertical)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Vertical size of video frame
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Vertical size of video frame
        * <HR>
        */
        gsCompChVerticalRes = 700,
        //! Alias for #gsCompChVerticalRes
        gsMpegVerticalRes = gsCompChVerticalRes,
        /**
        *       Size of picture   X (Horizontal)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Horizontal size of video frame
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Horizontal size of video frame
        * <HR>
        */
        gsCompChHorizontalRes,
        //! Alias for #gsCompChHorizontalRes
        gsMpegHorizontalRes = gsCompChHorizontalRes,
        /**
        *       Chroma type 4:0:0, 4:2:0, 4:2:2, 4:4:4
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - One of #GS_MPEG_CHROMA_FORMAT_420,
        * #GS_MPEG_CHROMA_FORMAT_422, #GS_MPEG_CHROMA_FORMAT_444
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - One of #GS_MPEG_CHROMA_FORMAT_420,
        * #GS_MPEG_CHROMA_FORMAT_422, #GS_MPEG_CHROMA_FORMAT_444
        * <HR>
        */
        gsCompChChromaFormat,
        //! Alias for #gsCompChChromaFormat
```

```
        gsMpegChromaFormat = gsCompChChromaFormat,
        /**
        *       DC Precision (mostly MPEG) 8..12
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_DC_PRECISION_8,
#GS_MPEG_DC_PRECISION_9,
        * #GS_MPEG_DC_PRECISION_10, #GS_MPEG_DC_PRECISION_11
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_DC_PRECISION_8,
#GS_MPEG_DC_PRECISION_9,
        * #GS_MPEG_DC_PRECISION_10, #GS_MPEG_DC_PRECISION_11
        * <HR>
        */
        gsCompChDCPrecision,
        //! Alias for #gsCompChDCPrecision
        gsMpegDCPrecision = gsCompChDCPrecision,
        /**
        *       Video signal aspect ratio
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - #GS_ASPECT_RATIO_SQUARE,
#GS_ASPECT_RATIO_4x3,
        * #GS_ASPECT_RATIO_16x9, #GS_ASPECT_RATIO_2_21x1
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - #GS_ASPECT_RATIO_SQUARE,
#GS_ASPECT_RATIO_4x3,
        * #GS_ASPECT_RATIO_16x9, #GS_ASPECT_RATIO_2_21x1
        * <HR>
        */
        gsCompChAspectRatio,
        //! Alias for #gsCompChAspectRatio
        gsMpegAspectRatio = gsCompChAspectRatio,
        /**
        *       MPEG file stream standard
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_STANDARD_SYSTEM,
#GS_MPEG_STANDARD_PROGRAM,
        * #GS_MPEG_STANDARD_TRANSPORT, #GS_MPEG_STANDARD_ELEMENTARY
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_STANDARD_SYSTEM,
#GS_MPEG_STANDARD_PROGRAM,
        * #GS_MPEG_STANDARD_TRANSPORT, #GS_MPEG_STANDARD_ELEMENTARY
        * <HR>
        */
        gsCompChStandard,
        //! Alias for #gsCompChStandard
        gsMpegStandard = gsCompChStandard,
```

```
        /**
        *   Video/Audio Language Code
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_LANGAUGE_ENGLISH, etc
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_LANGAUGE_ENGLISH, etc
        * <HR>
        */
        gsCompChLanguageCode,
        //! Alias for #gsCompChLanguageCode
        gsMpegLanguageCode = gsCompChLanguageCode,
        /**
        *       Closed Captioning Format
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_CC_FORMAT_CCUBE,
#GS_MPEG_CC_FORMAT_ATSC,
        * #GS_MPEG_CC_FORMAT_CCUBE_REORDER,
#GS_MPEG_CC_FORMAT_ATSC_REORDER
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - #GS_MPEG_CC_FORMAT_CCUBE,
#GS_MPEG_CC_FORMAT_ATSC,
        * #GS_MPEG_CC_FORMAT_CCUBE_REORDER,
#GS_MPEG_CC_FORMAT_ATSC_REORDER
        * <HR>
        */
        gsCompChCCFormat,
        //! Alias for #gsCompChCCFormat
        gsMpegCCFormat = gsCompChCCFormat,
        /**
        *       MPEG Concealment Vector
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - Not sure - Argus Encoder
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - Not sure - Argus Encoder
        * <HR>
        */
        gsCompChConcealmentVector,
        //! Alias for #gsCompChConcealmentVector
        gsMpegConcealmentVector = gsCompChConcealmentVector,
        /**
        *       Set encoding to closed gop or open gop
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - 0 = open gop, 1 = closed gop
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition -  0 = open gop, 1 = closed gop
```

```
* <HR>
*/
gsCompChClosedGop,
//! Alias for #gsCompChClosedGop
gsMpegClosedGop = gsCompChClosedGop,
/**
*       Set the next GOP start time code value
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Time code in frames (used def tctype)
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Time code in frames (used def tctype)
* <HR>
*/
gsCompChAdjustGopTC,
//! Alias for #gsCompChAdjustGopTC
gsMpegAdjustGopTC = gsCompChAdjustGopTC,
/**
*       Set MPEG encoder to use alternate co-efficent tables
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - 1/0
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - 1/0
* <HR>
*/
gsCompChAltCoEffTable,
//! Alias for #gsCompChAltCoEffTable
gsMpegAltCoEffTable = gsCompChAltCoEffTable,
/**
*       Set encoder to use non linear quantization
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - 1/0
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - 1/0
* <HR>
*/
gsCompChNonLinearQuant,
//! Alias for #gsCompChNonLinearQuant
gsMpegNonLinearQuant = gsCompChNonLinearQuant,
/**
*       Set the multiplexer (overall) bit rate
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Bits per second
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Bits per second
* <HR>
```

```
*/
gsCompChMuxRate,
//! Alias for #gsCompChMuxRate
gsMpegMuxRate = gsCompChMuxRate,
/**
*       Audio packet size
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Size of an audio packet in bytes
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Size of an audio packet in bytes
* <HR>
*/
gsCompChAudPacketSize,
//! Alias for #gsCompChAudPacketSize
gsMpegAudPacketSize = gsCompChAudPacketSize,
/**
*       Video packet size
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Size of a video packet in bytes
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Size of a video packet in bytes
* <HR>
*/
gsCompChVidPacketSize,
//! Alias for #gsCompChVidPacketSize
gsMpegVidPacketSize = gsCompChVidPacketSize,
/**
*       Stream ID for AUDIO 0xc0 (0x1c0)
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Audio Strean ID
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Audio Stream ID
* <HR>
*/
gsCompChAudioStreamID,
//! Alias for #gsCompChVideoStreamID
gsMpegAudioStreamID = gsCompChAudioStreamID,
/**
*       Stream ID for VIDEO 0xe0 (0x1e0)
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Video Stream ID
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Video Stream ID
* <HR>
*/
```

```
gsCompChVideoStreamID,
//! Alias for #gsCompChVideoStreamID
gsMpegVideoStreamID = gsCompChVideoStreamID,
/**
*       Program ID of the video stream within a transport container
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Video program id (PID)
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Video program id (PID)
* <HR>
*/
gsCompChAudioStreamPID,
//! Alias for #gsCompChAudioStreamPID
gsMpegAudioStreamPID = gsCompChAudioStreamPID,
/**
*       Program ID of the audio stream within a transport container
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - Audio program id (PID)
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Audio program id (PID)
* <HR>
*/
gsCompChVideoStreamPID,
//! Alias for #gsCompChVideoStreamPID
gsMpegVideoStreamPID = gsCompChVideoStreamPID,
/**
*       Allow settings to be changed.  Used to determine if settings
* can be changed (on the fly, without restart),
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - 0 disable changes, 1 allow changes
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - 0 disable changes, 1 allow changes
* <HR>
*/
gsCompChAllowSettings,
//! Alias for #gsCompChAllowSettings
gsMpegAllowSettings = gsCompChAllowSettings,
/**
* Fourcc code for compression.  Set one video or audio channel
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
```

```
gsCompChFourCC,
/**
* bit count for compression 8 / 10 / 24 / 32
* set to return bit count in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsCompChBitCount,
/**
* Size of each image in bytes
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsCompChSizeImage,
/**
* rate of frame
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsCompChRate,
/**
* Scale for frame rate
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsCompChScale,
/**
* Bytes per video line
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
```

```
*/
gsCompChPitch,
/**
* Encoding compression format ie avi mov dpx
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsVideoEncodeFormat,
/**
* Encoding compression format ie wav aiff
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsAudioEncodeFormat,
/**
* last compression change ms for updating hte clip bin
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsCompChannelChangeMs,
/**
* single link, dual link or alpha
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
* <HR>
*/
gsAlphaChromaSource,
/**
* RGBA BGRA ycbcr
* set to return fourcc code in dwPosition
* <BR>
* \li cmdType::ctSetValue - See fccDef.h for know fourcc
* <BR>
* \li cmdType::ctGetValue - See fccDef.h for know fourcc
```

```
* <HR>
*/
gsCompressionType,
/**
* <BR>
* \li cmdType::ctSetValue -
* <BR>
* \li cmdType::ctGetValue -
* <HR>
*/
gsVideoStandard,
/**
* Reset the channel to the new setup
* <BR>
* \li cmdType::ctSetValue -
* <BR>
* \li cmdType::ctGetValue -
* <HR>
*/
gsResetChannel,
/**
* <BR>
* \li cmdType::ctSetValue -
* <BR>
* \li cmdType::ctGetValue -
* <HR>
*/
gsEnableHDSDFormat,
/**
* Enable capture of vertical blank?
* <BR>
* \li cmdType::ctSetValue -
* <BR>
* \li cmdType::ctGetValue -
* <HR>
*/
gsVBlankEnable,
/**
* Enable/Disable LUTs
* <BR>
* \li cmdType::ctSetValue -
* <BR>
* \li cmdType::ctGetValue -
* <HR>
*/
gsLUTEnable,
/**
*
* <BR>
```

```
        * \li cmdType::ctSetValue -
        * <BR>
        * \li cmdType::ctGetValue -
        * <HR>
        */
        gsAudioFileType,
        /**
        *
        * <BR>
        * \li cmdType::ctSetValue -
        * <BR>
        * \li cmdType::ctGetValue -
        * <HR>
        */
        gsAudioBitSize,
        /**
        *
        * <BR>
        * \li cmdType::ctSetValue -
        * <BR>
        * \li cmdType::ctGetValue -
        * <HR>
        */
        gsAudioFrequency,


        /**
        * Channel Compression format
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition -
        * #GS_SIGFORM_CCIR_NTSC #GS_SIGFORM_CCIR_PAL
        * #GS_SIGFORM_1035i_30_260M #GS_SIGFORM_1035i_30X_260M
        * #GS_SIGFORM_1080i_30 #GS_SIGFORM_1080i_30X #GS_SIGFORM_1080i_25
#GS_SIGFORM_1080i_24 #GS_SIGFORM_1080i_24X
        * #GS_SIGFORM_1080_30 #GS_SIGFORM_1080_30X #GS_SIGFORM_1080_25
#GS_SIGFORM_1080_24 #GS_SIGFORM_1080_24X
        * #GS_SIGFORM_720_60 #GS_SIGFORM_720_60X #GS_SIGFORM_CUSTOM
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_SIGFORM_NTSC #GS_SIGFORM_PAL
        * #GS_SIGFORM_CCIR_NTSC #GS_SIGFORM_CCIR_PAL
        * #GS_SIGFORM_1035i_30_260M #GS_SIGFORM_1035i_30X_260M
        * #GS_SIGFORM_1080i_30 #GS_SIGFORM_1080i_30X #GS_SIGFORM_1080i_25
#GS_SIGFORM_1080i_24 #GS_SIGFORM_1080i_24X
        * #GS_SIGFORM_1080_30 #GS_SIGFORM_1080_30X #GS_SIGFORM_1080_25
#GS_SIGFORM_1080_24 #GS_SIGFORM_1080_24X
        * #GS_SIGFORM_720_60 #GS_SIGFORM_720_60X #GS_SIGFORM_CUSTOM
        * <BR>MEDIACMD::dwStart - Bit array of supported types
        * <HR>
```

```
        */

        gsSignalFormat = 900,  // NTSC CCIR HD 16x9 etc (dwPosition,
supported=dwStart)

        /**
        * Channel Compression format/type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition -
        * <BR>#GS_COMPTYPE_SOFTWARE    Software passed codec on main
processor
        * <BR>#GS_COMPTYPE_MJPEG Motion JPEG hardware codec (LSI, Zoran, C-
Cube, etc)
        * <BR>#GS_COMPTYPE_WAVELET Wavelet hardware codec
        * <BR>#GS_COMPTYPE_MPEG1 MPEG 1 hardware compatible codec
        * <BR>#GS_COMPTYPE_MPEG2 MPEG 2 hardware compatible codec
        * <BR>#GS_COMPTYPE_MPEG2I Editable MPEG 2 I Frame Only compatible
codec
        * <BR>#GS_COMPTYPE_MPEG2IBP      MPEG 2 long gop hardware compatible
codec
        * <BR>#GS_COMPTYPE_DV25 Hardware DV25, DVCPRO, DVCPRO25
        * <BR>#GS_COMPTYPE_DV50 Hardware DV50, DVCPRO50
        * <BR>#GS_COMPTYPE_DVSD Hardware Standard DV Bluebook, DVPRO, DVSD
        * <BR>#GS_COMPTYPE_DV100 High Def DV codec
        * <BR>#GS_COMPTYPE_HDPAN Panasonic HD to SDI codec
        * <BR>#GS_COMPTYPE_HDSONY Sony HD to SDI codec
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition -
        * <BR>#GS_COMPTYPE_SOFTWARE    Software passed codec on main
processor
        * <BR>#GS_COMPTYPE_MJPEG Motion JPEG hardware codec (LSI, Zoran, C-
Cube, etc)
        * <BR>#GS_COMPTYPE_WAVELET Wavelet hardware codec
        * <BR>#GS_COMPTYPE_MPEG1 MPEG 1 hardware compatible codec
        * <BR>#GS_COMPTYPE_MPEG2 MPEG 2 hardware compatible codec
        * <BR>#GS_COMPTYPE_MPEG2I Editable MPEG 2 I Frame Only compatible
codec
        * <BR>#GS_COMPTYPE_MPEG2IBP      MPEG 2 long gop hardware compatible
codec
        * <BR>#GS_COMPTYPE_DV25 Hardware DV25, DVCPRO, DVCPRO25
        * <BR>#GS_COMPTYPE_DV50 Hardware DV50, DVCPRO50
        * <BR>#GS_COMPTYPE_DVSD Hardware Standard DV Bluebook, DVPRO, DVSD
        * <BR>#GS_COMPTYPE_DV100 High Def DV codec
        * <BR>#GS_COMPTYPE_HDPAN Panasonic HD to SDI codec
        * <BR>#GS_COMPTYPE_HDSONY Sony HD to SDI codec
        * <BR>MEDIACMD::dwStart - Bit array of supported types
        * <HR>
        */
```

```
        gsCompType,                     // MJPG, MPEG2, Uncompressed
(dwPosition, supported=dwStart)

        /**
        * Compression setting by total throughput
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Size of compressed stream in kilobytes per
second
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Size of compressed stream in kilobytes per
second
        * <BR>MEDIACMD::dwStart - Smallest size possible
        * <BR>MEDIACMD::dwEnd - Largest size possible
        * <HR>
        */
        gsCompRateSize,                 // Total data throughput - frame size
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Compression setting by compression ratio
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Ratio * 100 (eg 2:1 = 200)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Ratio * 100 (eg 2:1 = 200)
        * <BR>MEDIACMD::dwStart - Smallest available ration * 100
        * <BR>MEDIACMD::dwEnd - Largest available ration * 100
        * <HR>
        */
        gsCompRateRatio,                // Total data rate - ratio * 100 (eg 2:1 = 200)
(dwPosition, min=dwStart, max=dwEnd)
        /**
        * Compression setting by compression percentage of original size
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Percentage * 100 (eg 50% compression =
5000)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Percentage * 100 (eg 50% compression =
5000)
        * <BR>MEDIACMD::dwStart - Smallest available percentage (usually 0)
        * <BR>MEDIACMD::dwEnd - Largest available percentage (usually 10000)
        * <HR>
        */
        gsCompRatePercent,              // Total data rate - precentage * 100 (0-10000)
of maximum (dwPosition, min=dwStart, max=dwEnd)
        /**
        * Number of frames per 'group of pictures'.  For MPEG compression as well as
        * defining keyframe interval for Cinepac, Indeo, MPEG-4, etc.
```

```
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Number of frames between keyframes of MPEG
'GOP' frame length
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Number of frames between keyframes of MPEG
'GOP' frame length
        * <BR>MEDIACMD::dwStart - Minimum possible size of group of pictures
(usually 0)
        * <BR>MEDIACMD::dwEnd - Largest possible size of group of pictures (up to
10000 for MPEG 4)
        * <HR>
        */
        gsCompGOPSize,                          // Number of elements in gop
        /**
        * Number of I Frame elements per GOP
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes available on drive
        * <HR>
        */
        gsCompIFactor,                  // Number of elements in gop
        /**
        * Number of B Frame elements per GOP
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes available on drive
        * <HR>
        */
        gsCompBFactor,                          // Number of elements in gop
        /**
        * Number of P Frame elements per GOP
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes available on drive
        * <HR>
        */
        gsCompPFactor,                          // Number of elements in gop
        /**
        * Refence period to determine amount and order of P and B frames
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
```

```
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes available on drive
        * <HR>
        */
        gsCompRefPeriod,                        // Number of elements in gop
        /**
        * Total storage available on current recording drive in kilobytes
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes available on drive
        * <HR>
        */
        gsTotalStorageAvail,    // Total available storage (dwPosition)
        /**
        * Total storage free on current recording drive in kilobytes
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Kilobytes free on drive
        * <HR>
        */
        gsTotalStorageFree,             // Total free storage (total free space/cur data
rate) (dwPosition)
        /**
        * Total recording time available on current recording drive at current
compression level
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Number of frames available to record to
        * <HR>
        */
        gsTotalTimeAvail,               // Total available time (dwPosition)
        /**
        * Total recording time free on current recording drive at current compression
level
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Number of frames free to record to
        * <HR>
        */
        gsTotalTimeFree,                // Total free time (total free space/cur data rate)
(dwPosition)
```

```
        /**
        * VTR emulation ID type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Any WORD VTR ID - See Control key in registry
docs and LocalConfig.exe
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Any WORD VTR ID - See Control key in registry
docs and LocalConfig.exe
        * <HR>
        */
        gsVtrType,                              // Emulation type (dwPosition)

        /**
        * Front panel/GUI Interface Local Mode
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - If 1 then local control available, else remote
only
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - If 1 then local control available, else remote
only
        * <HR>
        */
        gsLocal = 1000,                         // Local enable/disable (dwPosition)
        /**
        * Supported read/write file types
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Currently #GS_SUPFILE_AVI,
#GS_SUPFILE_ODML,
        * #GS_SUPFILE_QT, #GS_SUPFILE_OMFI, #GS_SUPFILE_FIX,
#GS_SUPFILE_AUDONLY,
        * #GS_SUPFILE_STILLS, #GS_SUPFILE_UNK, #GS_SUPFILE_ANY
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently #GS_SUPFILE_AVI,
#GS_SUPFILE_ODML,
        * #GS_SUPFILE_QT, #GS_SUPFILE_OMFI, #GS_SUPFILE_FIX,
#GS_SUPFILE_AUDONLY,
        * #GS_SUPFILE_STILLS, #GS_SUPFILE_UNK, #GS_SUPFILE_ANY
        * <BR>MEDIACMD::dwStart - Bit array of supported types per dwPosition above.
        * <HR>
        */
        gsSupportedFileTypes,  // In order of favorites (dwPosition)
        /**
        * File types for this channel to ignore
        * <BR>
        * \li cmdType::ctSetValue
```

```
        * <BR>MEDIACMD::dwPosition - Currently #GS_SUPFILE_AVI,
#GS_SUPFILE_ODML,
        * #GS_SUPFILE_QT, #GS_SUPFILE_OMFI, #GS_SUPFILE_FIX,
#GS_SUPFILE_AUDONLY,
        * #GS_SUPFILE_STILLS, #GS_SUPFILE_UNK, #GS_SUPFILE_ANY
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Currently #GS_SUPFILE_AVI,
#GS_SUPFILE_ODML,
        * #GS_SUPFILE_QT, #GS_SUPFILE_OMFI, #GS_SUPFILE_FIX,
#GS_SUPFILE_AUDONLY,
        * #GS_SUPFILE_STILLS, #GS_SUPFILE_UNK, #GS_SUPFILE_ANY
        * <HR>
        */
        gsIgnoreFileTypes,              // Per above     (dwPosition)
        /**
        * Disable recording on this channel or this channel does not support recording.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 1 to disable recording, or 0 to enable
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 1 to disable recording, or 0 to enable (play only
channels always return 1)
        * <HR>
        */
        gsRecInhibit,                   // Inhibit recording (dwPosition)
        /**
        * Select recording drive
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Bit representing drive where 0=C:, 1=D: etc
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Bit representing drive where 0=C:, 1=D: etc
        * <BR>MEDIACMD::dwStart - Bit array of available drives
        * <HR>
        */
        gsRecDrive,                             // Record drives (dwPosition,
available=dwStart)
        /**
        * Change the defual record filename
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::arbID -  New next record filename
        * <BR>  MEDIACMD::cfFlags - must be set to cfUseClipID
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - gsTrue/gsFalse
        * <BR>MEDIACMD::arbID -  Next record filename
        * <BR>  MEDIACMD::cfFlags - must be set to cfUseClipID
        * <HR>
        */
```

```
        gsRecFileName,                  // Record drives (dwPosition,
available=dwStart)
        /**
        * Recording rate by throughput in kilobytes per second
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Target size of recorded stream in kilobytes per
second
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Target size of recorded stream in kilobytes per
second
        * <BR>MEDIACMD::dwStart - Smallest size possible
        * <BR>MEDIACMD::dwEnd - Largest size possible
        * <HR>
        */
        gsRecRate,                      // Default recording rate (dwPosition,
default=dwStart)
        /**
        * Default video/stream record file type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * <HR>
        */
        gsRecFileFormat,            // Type of file to record SEE suppfiletypes
(dwPosition, avail=dwStart)
        /**
        * Default audio record file type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * <HR>
        */
        gsRecAudFileFormat,         // Type of audio file, or -1 if embedded
(dwPosition, avail=dwStart)
        /**
        * Disable file deletion on this channel
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 1 to disable delete command, or 0 to enable
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 1 to disable delete command, or 0 to enable
        * <HR>
        */
        gsDelInhibit,               // Inhibit deletion (dwPosition)
```

```
        /**
        * Allows/Inhiibits clips being Deleted from Bin or TC Space
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - TRUE/FALSE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - TRUE/FALSE
        * <HR>
        */
        gsInsInhibit,                   // Inhibit insertion of clips (dwPosition)
        /**
        * Allows/Inhiibits clips being added to Bin or TC Space
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition - TRUE/FALSE
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - TRUE/FALSE
        * <BR>
        */
        gsConvertFileFormat,    // Type of file to convert to (dwPosition, avail = dwStart)
        /**
        * Default audio conversion file type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Uses mftXXX enum from MediaReactorTypes.h
        * <HR>
        */
        gsConvertAudFileFormat,         // Type of audio file, or -1 if embedded
(dwPosition, avail = dwStart)
        /**
        * Default length, in frames, for a still graphics file being added as a clip
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Duration in frames
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Duration in frames
        * <HR>
        */
        gsDefStillLen,                  // Default still length (dwPosition)
        /**
        * Current reference system time for house VITC or house LTC if available, if
        * not then from system clock interpolated with performanace counter.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Current time code position
        * <BR>MEDIACMD::dwStart - Current milliseconds position
        * <BR>MEDIACMD::dwEnd - Current date
```

```
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current time code position
        * <BR>MEDIACMD::dwStart - Current milliseconds position
        * <BR>MEDIACMD::dwEnd - Current date
        * <HR>
        */
        gsSysTime,                          // System Reference time (dwPostion,
dwStart, dwEnd)
        /**
        * Current reference system time for house VITC or house LTC if available, if
        * not then from system clock interpolated with performanace counter.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Current dysnc milliseconds
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current dysnc milliseconds
        * <HR>
        */
        gsDSyncMs,                          // DSync MS counter for current channel
        /**
        * Current hardware port used by channel.  Mostly for COMx: port
        * selection of CTL and EXT channels.
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - New com port
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current com port
        * <BR>MEDIACMD::dwStart - Available com ports as bit array
        * <HR>
        */
        gsHwPort,                           // Currently only for ext drv (dwPostion
= current, dwStart = available)
        /**
        * Playback only output or allow edit to edit
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_PBEE_AUTO (playback or e to e),
#GS_PBEE_PB (playback only)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_PBEE_AUTO (playback or e to e),
#GS_PBEE_PB (playback only), #GS_PBEE_DEFAULT (device default read only)
        * <BR>MEDIACMD::dwStart - Bit array of available commands per dwPosition
settings above
        * <HR>
        */
        gsPBEE,                                     // dwPosition
        /**
        * Video reference for servo select
        * <BR>
```

```
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_SERVOREF_AUTO (ext is avail, else int),
#GS_SERVOREF_EXT (always external)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_SERVOREF_AUTO (ext is avail, else int),
#GS_SERVOREF_EXT (always external), #GS_SERVOREF_DEFAULT (device default read
only)
        * <BR>MEDIACMD::dwStart - Bit array of available commands per dwPosition
settings above
        * <HR>
        */
        gsServoRefSelect,               // dwPosition
        /**
        * Head select
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_HEADSEL_RECPLAY,
#GS_HEADSEL_PLAY
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_HEADSEL_RECPLAY,
#GS_HEADSEL_PLAY, GS_HEADSEL_DEFAULT (device default read only)
        * <BR>MEDIACMD::dwStart - Bit array of available commands per dwPosition
settings above
        * <HR>
        */
        gsHeadSelect,                   // dwPosition
        /**
        * Colour frame select
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_CLRFRM_2FLD, #GS_CLRFRM_4FLD,
#GS_CLRFRM_8FLD
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_CLRFRM_2FLD, #GS_CLRFRM_4FLD,
#GS_CLRFRM_8FLD, GS_CLRFRM_DEFAULT
        * <BR>MEDIACMD::dwStart - Bit array of available commands per dwPosition
settings above
        * <HR>
        */
        gsColorFrame,                   // dwPosition
        /**
        * Video reference disable
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - GS_VIDREF_DISABLE, GS_VIDREF_ENABLE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - GS_VIDREF_DISABLE, GS_VIDREF_ENABLE
        * <BR>MEDIACMD::dwStart - Bit array of available commands per dwPosition
settings above
```

```
* <HR>
*/
gsVidRefDisable,                    // dwPosition
/**
* Get Play count delay for the VTR interp
* <BR>
* \li cmdType::ctSetValue
* <BR>Not Supported
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition stores the value 7 for default
* <HR>
*/
gsPlayCountDelay,                   //dwPoisition     1022
/**
* Use fake edit mode for MPEG bumping.  Basically, all non
* play speed commands will be emulated, and once a play (lock)
* is reached the card will be synced to that time and play.
* Dangerous if sync does not happen quickly...
* <BR>
* \li cmdType::ctSetValue
* <BR>MEDIACMD::dwPosition 1 turns on fake edit, 0 turns off
* \li cmdType::ctGetValue
* <BR>MEDIACMD::dwPosition 1 fake edit on, 0 fake edit of
* <HR>
*/
gsEmulateEditBumping,
/**
* Special command alt value for position requests, goes
* to key frame nearest to requested frame
* <BR>
* \li cmdType::ctPause, cmdType::ctPlay
* <BR>MEDIACMD::dwPosition Target Position
* <HR>
*/
cmdaltNearestKeyFrame,
/**
* Special command alt value for position requests, goes
* to key frame after to requested frame
* <BR>
* \li cmdType::ctPause, cmdType::ctPlay
* <BR>MEDIACMD::dwPosition Target Position
* <HR>
*/
cmdaltNextKeyFrame,
/**
* Special command alt value for position requests, goes
* to key frame before to requested frame
* <BR>
* \li cmdType::ctPause, cmdType::ctPlay
```

```
        * <BR>MEDIACMD::dwPosition Target Position
        * <HR>
        */
        cmdaltPrevKeyFrame,
        /**
        * Special command alt value for position requests, goes
        * to first frame of actual (non-black) video after the requested frame
        * <BR>
        * \li cmdType::ctPause, cmdType::ctPlay
        * <BR>MEDIACMD::dwPosition Target Position
        * <HR>
        */
        cmdaltStartOfMessage,
        /**
        * Special command is video input valid
        * GS_TRUE / GS_FALSE
        * <BR>
        * \li cmdType::ctGetValue, cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition  -True / False is input Valid
        * <BR>  MEDIACMD::dwStart     -signal format of the input
        * <HR>
        */
        gsVidInputValid,
        /**
        * Special command is genlock input valid
        * GS_TRUE / GS_FALSE
        * <BR>
        * \li cmdType::ctGetValue, cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition  -True / False is genlock Valid
        * <BR>  MEDIACMD::dwStart     -signal format of the genlock signal
        * <HR>
        */
        gsVidGenlockValid,
        /**
        * Special command to set/get edit mode for slow MPEG boards
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition  - #GS_SERIALEDITMODE_NONE,
#GS_SERIALEDITMODE_IGNORE, #GS_SERIALEDITMODE_FAKE
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition  - #GS_SERIALEDITMODE_NONE,
#GS_SERIALEDITMODE_IGNORE, #GS_SERIALEDITMODE_FAKE
        * <HR>
        */
        gsSerialEditMode,


#if 0   // Reserved
        /**
```

```
        *
        */
        gsChannelAdd = 2000,
        gsChannelDel,
        gsChannelAddress,
        gsChannelPort,
        gsChannelTarget,
        gsChannelType,
        gsChannelEnable,
#endif

        /**
        * Enable/Disable/Flush error log
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 1 Enable, 0 Disable, -1 Flush
        * <BR>MEDIACMD::dwStart - Start messages
        * <BR>MEDIACMD::dwEnd - Maximum messages
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 1 Enable, 0 Disable
        * <BR>MEDIACMD::dwStart - Total messages
        * <BR>MEDIACMD::dwEnd - Maximum messages
        * <HR>
        */
        gsErrorLog = 10000,
        /**
        * Get/Set Error Log Name
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::arbID - New Log Name
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Current Log Name
        * <HR>
        */
        gsErrorLogName,
        /**
        * Gets the starting ms value in message units
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - First message ms (ever)
        * <HR>
        */
        gsErrorLogStartMs,
        /**
        * Gets current ms time per log entries (for relative and date absolute)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Current ms
        * <HR>
        */
        gsErrorLogCurrentMs,
        /**
```

```
        * Gets the last change value, modified with each new entry.
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Last change value
        * <BR>MEDIACMD::dwStart - First available message number
        */
        gsErrorLogLastChange,
        /**
        * Gets/Set an error message to/from the log
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Send -1 for first num, send last returned
number to get next, Return num
        * <BR>MEDIACMD::lSpeed - Send format type
        * <BR>MEDIACMD::dwStart - Error Code
        * <BR>MEDIACMD::dwCmdAlt - Time of message ms
        * <BR>MEDIACMD::cfFlags - cfPreview causes RAW return
        * <BR>MEDIACMD::dwStart - First available message number
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwStart - Error Code
        * <BR>MEDIACMD::arbID - Raw message
        */
        gsErrorLogMessage,

        /**
        * Buffer levels for playback/record (real time)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> Not supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Buffers stored in board
        * <BR>MEDIACMD::dwStart - Buffers in queue to board
        * <BR>MEDIACMD::dwEnd - Buffers in queue from disk
        * <BR>. MEDIACMD:;dwInfoChannels - total system buffers available in board +
memory / system frame size
        * <BR>MEDIACMD::dwVideoChannels - Selected channel for buffer request
        * <BR>MEDIACMD::dwAudioChannels - Selected channel for buffer request
        * <HR>
        */
        gsSysBufferLevel = 20000,
        /**
        * Memory usage
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> Not supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Memory we are using
        * <BR>MEDIACMD::dwStart - Memory used in system
        * <BR>MEDIACMD::dwEnd - Total memory in system
        * <HR>
        */
```

```
        gsSysMemoryUsage,
        /**
        * CPU usage
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> Not supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - CPU we are using
        * <BR>MEDIACMD::dwStart - CPU used in system
        * <BR>MEDIACMD::dwEnd - Portion of CPU in kernel mode
        * <HR>
        */
        gsSysCPUUsage,
        /**
        * Dropped frames
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> Not supported
        * \li cmdType::ctGetValue
   * <BR>    MEDIACMD::dwPosition - Last drop number (includes off speed play that
should drop)
   * <BR>    MEDIACMD::lSpeed - millisecond time of last drop
   * <BR>    MEDIACMD::dwStart - Total playback dropped (since first run)
   * <BR>    MEDIACMD::dwEnd - Total record dropped (since first run)
        * <HR>
        */
        gsDroppedFrames,

        /**
        * Sets/Returns auto proxy mode
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Proxy mode 0=disabled,
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Proxy mode 0=disabled,
        */
        gsProxyMode = 50000,
        /**
        * Returns the status of any proxy generation
        * <BR>
        * \li MEDIACMD::ctCmd -
        * \li cmdType::ctPlay - creating a proxy from file on disk
        * \li cmdType::ctRecord - creating a proxy from a recording file
        * \li MEDIACMD::dwCmdAlt - Last change in list ms
        * \li MEDIACMD::dwPosition - Current encode frame
        * \li MEDIACMD::dwEnd - current length of file
        * \li MEDIACMD::dwStart - current processor percentage
        * \li MEDIACMD::arbID - current encode path + name
        */
```

```
gsProxyStatus,
/**
* Return the next proxy source file name.  If the previous name
* is set to NULL then return the first clip in the list.
* <BR>
* \li cmdType::ctSetValue
* <BR>  - not supported
* \li cmdType::ctGetValue
* <BR>  -in-  MEDIACMD::arbID - Last returned file name or NULL
* <BR>  -out- MEDIACMD::arbID - Next file name in proxy list
* <BR>  MEDIACMD::dwPosition - Not used yet
* <BR>  MEDIACMD::dwStart - Not used yet
* <BR>  MEDIACMD::dwEnd - Not used yet
* <HR>
*/
gsGetNextProxy,
/**
* Add a new proxy file to the list
* <BR>
* \li MEDIACMD::arbID - File name to proxy
* \li MEDIACMD::dwStart - Start frame of proxy
* \li MEDIACMD::dwEnd - End frame of proxy
*/
gsAddProxy,
/**
* Set a proxy file to next in list
* <BR>
* \li MEDIACMD::arbID - File to promote
*/
gsPromoteProxy,
/**
* Remove a proxy file from the list
* <BR>
* \li MEDIACMD::arbID - File to remove
*/
gsRemoveProxy,
/**
* Get Set the max cpu percentage
* <BR>
* \li cmdType::ctSetValue
* <BR> MEDIACMD::dwPosition - 0 No CPU limit, 1..100 max cpu usage
* \li cmdType::ctGetValue
* <BR> MEDIACMD::dwPosition - Current cpu usage
* <BR> MEDIACMD::dwStart - Current MAX cpu useage
* <BR> MEDIACMD::dwEnd - Default CPU Max Usage
*/
gsProxyCPUUsage,

/**
```

```
        * Get VVW version number
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with version number
        * <HR>
        */
        gsVVWVersion = 60000,          // VVW Version (dwPosition, arbID = string)
        /**
        * Get MediaReactor version number
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with version number
        * <HR>
        */
        gsMEVersion,                   // Media Reactor Version  (dwPosition, arbID =
string)
        /**
        * Get VVW type description
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with VVW machine type
        * <HR>
        */
        gsVVWType,                     // Name of vvw model (dwPosition,
arbID = string)
        /**
        * Get VVW channel type
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with channel type
        * <HR>
        */
        gsVVWChannelType,              // Description of channel (dwPosition, arbID =
string)
        /**
        * Get/Set VVW channel name
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 0 Get Current, 1 Get Default
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with desired channel
name
```

```
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::arbID - Zero terminated ansi string with current channel
name
        * <HR>
        */
        gsVVWChannelName,           // Name of channel (dwPosition, arbID = string)
        /**
        * Get VVW Licenst status
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not supported
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - License type (-1=invalid, 0=perm, 1=days,
2=runs, 3=users)
        * <BR>MEDIACMD::dwStart - 0 for perm, # for days/runs/users
        * <BR>  MEDIACMD::dwEnd - License flages (app dependant)
        * <BR>  MEDIACMD::lSpeed - License level (app dependant)
        * <BR>MEDIACMD::arbID - Zero terminated ansi string about current license
        * <HR>
        */
        gsVVWLicense,               // Status of license


        /**
        * Setup on screen monitor (VGA Monitor)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - 1 to enable, 0 to disable
        * <BR>MEDIACMD::dwEnd - Left corner
        * <BR>MEDIACMD::dwStart - Top corner
        * <BR>MEDIACMD::dwCmdAlt - Size (0 = Default, 1 = Full, 2 = Half, 3 =
Quarter)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 1 to enable, 0 to disable
        * <BR>MEDIACMD::dwEnd - Left corner
        * <BR>MEDIACMD::dwStart - Top corner
        * <BR>MEDIACMD::dwCmdAlt - Size (0 = Default, 1 = Full, 2 = Half, 3 =
Quarter)
        * <HR>
        */
        gsMonitor = 64000,          // Set/Get info on VGA or Secondary NTSC
monitor
        /**
        * Set handles to windows
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Handle to target window or -1
        * <BR>MEDIACMD::dwEnd - Handle to owner application window
        * <BR>MEDIACMD::dwStart - Window flags
```

```
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Handle to target window
        * <BR>MEDIACMD::dwEnd - Handle to owner application window or -1
        * <BR>MEDIACMD::dwStart - Window flags
        * <HR>
        */
        gsMonitorHwnds,
        //! Alias for #gsMonitorHwnds for older apps
        gsHwnds = gsMonitorHwnds,                                   // Get above info
        /**
        * Turns VGA display on / off without killing to he window
        * Can use this later to set refresh rates - aspect ratios or what not
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - GS_TRUE, GS_FALSE
        * <BR>MEDIACMD::dwEnd -
        * <BR>MEDIACMD::dwStart -
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - GS_TRUE, GS_FALSE
        * <BR>MEDIACMD::dwEnd -
        * <BR>MEDIACMD::dwStart
        * <HR>
        */
        gsMonitorDisplay,
        /**
        *   Get a capture of the current output (input passthrough or
        * current clip output).  Use #cmdType::ctGetValue to get a preview.
        *<BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Type of capture that is going to be used
        * <BR>MEDIACMD::arbID - Optional, depends on dwPosition
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Preview type #GS_MONITORGRAB_NONE,
        * #GS_MONITORGRAB_TYPE_BMP, #GS_MONITORGRAB_TYPE_JPG,
#GS_MONITORGRAB_SIZE_FULL,
        * #GS_MONITORGRAB_SIZE_HALF, #GS_MONITORGRAB_SIZE_QUARTER,
#GS_MONITORGRAB_TO_MEMORY,
        * #GS_MONITORGRAB_TO_UNC_PATH, #GS_MONITORGRAB_TO_HTTP,
#GS_MONITORGRAB_TO_NETWORK
        * <BR>MEDIACMD::arbID - Image data, if returned not saved
        * <BR>
        * To create a grab type, combine on type (bmp, jpg) with a size
        * (full, half, quarter) and a target (memory, path, http, network). <BR>
        * Depending on target, the arbID member will be filled in as follows:
        * \li GS_MONITORGRAB_TO_MEMORY - arbID not used
        * \li GS_MONITORGRAB_TO_UNC_PATH - arbID unified naming conventions
(UNC) path
        * \li GS_MONITORGRAB_TO_HTTP - arbID contains the name
        * \li GS_MONITORGRAB_TO_NETWORK - Not implemented
```

```
        * <HR>
        */
        gsMonitorGrab,
        /**
        * Get/Set a pointer to the Utility Monitor DTDraw class
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Pointer in DWORD to DTDraw class (always
RGB32)
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Pointer in DWORD to DTDraw class (always
RGB32)
        * <BR>
        * <HR>
        */
        gsUtilityMonitorDraw,
        /**
        * Get/Set the layout of the utility monitor
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Main layout of screen
        *       // Normal quad screen<BR>
        *       #define AVUM_CONFIG_QUAD_SPLIT                  0<BR>
        *       // One full screen on left, 3 1/4 on right<BR>
        *       #define AVUM_CONFIG_ONELEFT_THREERIGHT          1<BR>
        *       // One full screen on top, 3 1/2 on bottom<BR>
        *       #define AVUM_CONFIG_ONETOP_THREEBOTTOM          2<BR>
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Main layout of screen
        * <BR>
        * <HR>
        */
        gsUtilityMonitorDrawSetup,
        /**
        * Get/Set a waveform, vectorscope, etc on DTDraw, VGA output or Main output
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - Pointer to the DTDraw structure to draw on
        * <BR>  MEDIACMD::dwStart - Enable or disable  #GS_ENABLE #GS_DISABLE
        * <BR>  MEDIACMD::dwEnd - Target #GS_WAVEVECTOR_TARGET_DTDRAW,
#GS_WAVEVECTOR_TARGET_VGA, #GS_WAVEVECTOR_TARGET_OUTPUT
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Returns the DTDraw structure it is drawing on
        * <BR>  MEDIACMD::dwStart - Is enabled or disabled  #GS_ENABLE
#GS_DISABLE
        * <BR>  MEDIACMD::dwEnd - Target #GS_WAVEVECTOR_TARGET_DTDRAW,
#GS_WAVEVECTOR_TARGET_VGA, #GS_WAVEVECTOR_TARGET_OUTPUT
```

```
        * <HR>
        */
        gsWaveVectorSetup,
        /**
        * Get/Set type of waveform vector
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition - Waveform vector type(s)
#GS_WAVEVECTOR_PICTURE, #GS_WAVEVECTOR_VECTORSCOPE,
#GS_WAVEVECTOR_WAVEFORM
        * <BR>  MEDIACMD::dwStart - Channels to enable
#GS_WAVEVECTOR_CHANNEL_R, #GS_WAVEVECTOR_CHANNEL_G,
#GS_WAVEVECTOR_CHANNEL_B, #GS_WAVEVECTOR_CHANNEL_A,
#GS_WAVEVECTOR_CHANNEL_Y. #GS_WAVEVECTOR_CHANNEL_CR,
#GS_WAVEVECTOR_CHANNEL_CB
        * <BR>  MEDIACMD::dwEnd -
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Waveform vector type(s)
#GS_WAVEVECTOR_PICTURE, #GS_WAVEVECTOR_VECTORSCOPE,
#GS_WAVEVECTOR_WAVEFORM
        * <BR>  MEDIACMD::dwStart - Channels to enable
#GS_WAVEVECTOR_CHANNEL_R, #GS_WAVEVECTOR_CHANNEL_G,
#GS_WAVEVECTOR_CHANNEL_B, #GS_WAVEVECTOR_CHANNEL_A,
#GS_WAVEVECTOR_CHANNEL_Y. #GS_WAVEVECTOR_CHANNEL_CR,
#GS_WAVEVECTOR_CHANNEL_CB
        * <BR>  MEDIACMD::dwEnd
        * <HR>
        */
        gsWaveVectorType,
        /**
        * Get/Set area to use as source
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>  MEDIACMD::dwPosition -
        * <BR>  MEDIACMD::dwStart - Start line 0..(height-1)
        * <BR>  MEDIACMD::dwEnd - End line 1..height
        * <BR>  MEDIACMD::dwVideoChannels - Start pixel 0..(width-1) - not supported
yet
        * <BR>  MEDIACMD::dwAudioChannels - End pixel 1..width - not supported yet
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition -
        * <BR>  MEDIACMD::dwStart - Start line 0..(height-1)
        * <BR>  MEDIACMD::dwEnd - End line 1..height
        * <BR>  MEDIACMD::dwVideoChannels - Start pixel 0..(width-1) - not supported
yet
        * <BR>  MEDIACMD::dwAudioChannels - End pixel 1..width - not supported yet
        * <HR>
        */
        gsWaveVectorArea,
```

```
        /**
        * Get last update time in milliseconds
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>  MEDIACMD::dwPosition - Time of last update in milliseconds
        */
        gsWaveVectorLastChangeMs,

        /** Not to be used.  See VVWXMLNextDirEntry and VVWXMLFileInfo
        */
        gsDirGetList = 64250,
        /** Not to be used.  See VVWXMLNextDirEntry and VVWXMLFileInfo
        */
        gsDirGetInfo,
        /** Not to be used.  See VVWXMLNextDirEntry and VVWXMLFileInfo
        */
        gsDirGetFileInfo,
        /** Not to be used.  See VVWXMLNextDirEntry and VVWXMLFileInfo
        */
        gsDirGetFileGrab,

        /**
        * Check if channels exist
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwVideoChannels - Possible Video Channels
        * <BR>MEDIACMD::dwAudioChannels - Possible Audio Channels
        * <BR>MEDIACMD::dwInfoChannels - Possible Info Channels
        * <HR>
        */
        gsChannelsExist = 65536,// Do the channels exist (dwPosition - used
dwvid/aud/inf channels)
        /**
        * Get/Set clip mode state (else time code mode)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_CLIPMODE_CLIPSPACE,
#GS_CLIPMODE_TCSPACE, #GS_CLIPMODE_SINGLE, #GS_CLIPMODE_FILM current
types
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_CLIPMODE_CLIPSPACE,
#GS_CLIPMODE_TCSPACE, #GS_CLIPMODE_SINGLE, #GS_CLIPMODE_FILM current
types
        * <HR>
        */
        gsClipMode,                             // Are we in clip or timecode space mode
(dwPosition)
```

```
        /**
        * Get/Set record offset for VVW3x00 replay mode
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - Time code offset or 0 to reset
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Time code offset or 0 to reset
        * <HR>
        */
        gsRecOffset,                    // Mostly for multi digisuite records
        /**
        * Get channel capabilities
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - Bitwise array #GS_CHANCAP_PLAY,
#GS_CHANCAP_REVPLAY,
        * #GS_CHANCAP_PAUSE, #GS_CHANCAP_JOG, #GS_CHANCAP_SHUTTLE,
#GS_CHANCAP_SEEK,
        * #GS_CHANCAP_PREVIEW, #GS_CHANCAP_STOP, #GS_CHANCAP_ETOE,
#GS_CHANCAP_RECORD,
        * #GS_CHANCAP_EDIT, #GS_CHANCAP_RECSTOP,
#GS_CHANCAP_SELECTPRESET,
        * #GS_CHANCAP_EJECT, #GS_CHANCAP_LOOP, #GS_CHANCAP_VGAPREVIEW,
#GS_CHANCAP_AUDPREVIEW,
        * #GS_CHANCAP_FILE, #GS_CHANCAP_NET, #GS_CHANCAP_CLIPSPACE,
        * #GS_CHANCAP_TCSPACE, #GS_CHANCAP_ALL
        * <BR>
        * <HR>
        */
        gsChanCapabilities,             // Return capabilities of vvw channel
        /**
        * Get last change millisecond time from clip space, tc space or file
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - last change in ms aligned with Dsync
        * <HR>
        */
        gsLastChangeMs,                 // Are we in clip or timecode
space mode (dwPosition)
        /**
        * Get the state of the GPI ins, reset them with set
        * <BR>
        * \li cmdType::ctSetValue - reset in events to nothing
        * <BR>MEDIACMD::dwVideoChannels - GPI A-H (0-7  / 1-8)
        * <BR>MEDIACMD::dwAudioChannels - GPI I-P (8-15 / 9-16)
```

```
        * <BR>MEDIACMD::dwInfoChannels -  GPI Q-X (16-23/ 17-24)
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwVideoChannels - GPI A-H (0-7  / 1-8)
        * <BR>MEDIACMD::dwAudioChannels - GPI I-P (8-15 / 9-16)
        * <BR>MEDIACMD::dwInfoChannels -  GPI Q-X (16-23/ 17-24)
        * <BR>MEDIACMD::dwPosition - last change in ms aligned with Dsync
        * <HR>
        */
        gsGPIIn,                                        // Get/Set GPI inputs
        /**
        * Get the state of the GPI outs,
        * <BR>
        * \li cmdType::ctSetValue - set the GPIs up or down or pulse
        * <BR>MEDIACMD::dwVideoChannels - GPI A-H (0-7  / 1-8)
        * <BR>MEDIACMD::dwAudioChannels - GPI I-P (8-15 / 9-16)
        * <BR>MEDIACMD::dwInfoChannels -  GPI Q-X (16-23/ 17-24)
        * <BR>MEDIACMD::dwPosition - Type of set (0=low, 1=high, 2=pulse)
        * \li cmdType::ctGetValue - get the current gpi output state.
        * <BR>MEDIACMD::dwVideoChannels - GPI A-H (0-7  / 1-8)
        * <BR>MEDIACMD::dwAudioChannels - GPI I-P (8-15 / 9-16)
        * <BR>MEDIACMD::dwInfoChannels -  GPI Q-X (16-23/ 17-24)
        * <BR>MEDIACMD::dwPosition - Last Change Ms
        * <BR>A 1 in the GPI bitwise array means on or triggered, 0
        * is down or off.
        * <HR>
        */
        gsGPIOut,                                       // Get/Set GPI outputs
        /**
        * Get the current millisecond counter on the maching (NOT aligned ot anything).
        * NOTE:  This is handled directly for network channels on server side
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - the current millisecond counter
        * <HR>
        */
        gsCurrentMs,                                    // Current ms (dwPosition)
        /**
        * Get/Set whether we are adding on minute of black to start and end of each clip
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE, #GS_FALSE
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - #GS_TRUE, #GS_FALSE
        * <HR>
        */
        gsClipModePreroll,                              // Are we adding black around
the clips
```

```
        /**
        * Save the current parameters to the registry
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>No Parameters
        * \li cmdType::ctGetValue
        * <BR>Not Used
        * <HR>
        */
        gsSaveCurrent = 100000,         // Save the current params
        /**
        * Load a new clip space (or new if file does not exist)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - File name of new or existing clip space
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID - returns current file name of clip space
        * <HR>
        */
        gsLoadClipSpace,          // Load a new clip space
        /**
        * Load a new tc space (or new if file does not exist)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - File name of new or existing tc space
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID - returns current file name of tc space
        * <HR>
        */
        gsLoadTCSpace,          // Load a new ::VTR_TC space
        /**
        * Load a new film (or new if file does not exist)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - File name of new or existing film
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID - returns current file name of film
        * <HR>
        */
        gsLoadFilmSpace,         // Load a new ::Film space

        /**
        * Log in user (must have rights on the local machine)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - User name zero terminated followed by password
zero terminated.
```

```
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID - Returns user name zero terminated.
        * <HR>
        */
        gsUserLogIn = 900000,
        /**
        * Last change in user status
        * <BR>
        * \li cmdType::ctSetValue
        * Not Supported
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition - Last change in users (status or number)
        * <HR>
        */
        gsUserLastChangeMs,
        /**
        * Return a list of currently logged in users
        * <BR>
        * \li cmdType::ctSetValue
        * Not Supported
        * \li cmdType::ctGetValue
        * <BR>MEDIACMD::dwPosition - 0..max user
        * <BR>  MEDIACMD::arbID - User name zero terminated, location info zero
terminated
        * <BR>MEDIACMD::dwStart - User rights
        * NULL when all users have been returnd
        * <HR>
        */
        gsUserList,
        /**
        * Allow a user access to the unit - ONLY AVAILABLE ON LOCAL MACHINE
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - User name zero terminated, password zero
terminated
        * <BR> MEDIACMD::dwStart - User Rights
        * \li cmdType::ctGetValue
        * Not Supported
        * <HR>
        */
        gsUserAdd,
        /**
        * Remove a users access to the unit - ONLY AVAILABLE ON LOCAL MACHINE
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - User name zero terminated.
        * \li cmdType::ctGetValue
        * Not Supported
        * <HR>
```

```
        */
        gsUserDel,
        /**
        * Change a users rights - ONLY AVAILABLE ON LOCAL MACHINE
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - User name zero terminated.
        * <BR> MEDIACMD::dwPosition - the user rights
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID - User name zero terminated (current if NULL)
        * <BR> MEDIACMD::dwPosition - the user rights
        * <HR>
        */
        gsUserRights,
        /**
        * Change current users password (must be logged in)
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID - new password zero terminated
        * new password zero terminated.
        * \li cmdType::ctGetValue
        * Not Supported
        * <HR>
        */
        gsUserPasswd,


        /**
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition              - frame number
        * <BR> MEDIACMD::arbID                   - file and directory
        *
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition              - (GS_TRUE | GS_FALSE) is set
picon
        * <BR> MEDIACMD::arbID                   - file and directory
        */
        gsPiconFrame = 1000000,
        /**
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition              - frame number
        * <BR> MEDIACMD::arbID                   - file and directory
        *
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition              - (GS_TRUE | GS_FALSE) is set
jpeg
        * <BR> MEDIACMD::arbID                   - file and directory
```

```
        */
        gsJpegFrame,
        /**
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::arbID                  - directory to use (NULL = Use
Network Directory)
        *
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::arbID                  - directory using for images
        */
        gsImageDirectory,
        /**
        * <BR>
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition             - IN = frame number, OUT =
Width
        * <BR> MEDIACMD::dwStart          - Height
        * <BR> MEDIACMD::dwEnd       - Bits
        * <BR> MEDIACMD::arbID                  - Encoding
        * <BR> return frame number
        */
        gsFrameInfo,
        /**
        * <BR>
        * \li cmdType::ctSetValue
        * <BR> MEDIACMD::dwPosition             - frame number
        * <BR> MEDIACMD::dwStart          - Format (original, RGBA)
        * <BR> MEDIACMD::dwEnd       - Frame Size
        * <BR> MEDIACMD::arbID                  - Image Data
        *
        * \li cmdType::ctGetValue
        * <BR> MEDIACMD::dwPosition             - frame number
        * <BR> MEDIACMD::dwStart          - Format (original, RGBA)
        * <BR> MEDIACMD::dwEnd       - Frame Size
        * <BR> MEDIACMD::arbID                  - Image Data
        */
        gsRawFrame,

        /**
        * Service
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>
        * <HR>
        */
        gsVVWService  = 1100000,
```

```
        /**
        * For clip copy and translation lists set removes an item get returns the list like a
cliplist
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>
        * <HR>
        */
        gsInsertQueue,
        /**
        * For clip copy queue manipulation
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>
        * <HR>
        */
        gsXlatQueue,

        /**
        * Set the rate and scale in an XML file for later opening
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>
        * <HR>
        */
        gsXMLRateScale,

        /**
        * Set the rate and scale in an XML file for later opening
        * <BR>
        * \li cmdType::ctSetValue
        * <BR>
        * \li cmdType::ctGetValue
        * <BR>
        * <HR>
        */
        gsXMLFileProperties,

        //
        // Internal Properties.  Do not add to Java/VB/HTTP/etc
        //
```

```
        /** Get the optimal offset for a video frame to allow a header to be added
        */
        gsInternalGetImageOffset = 0xFFFFFFFDUL
};      // cmdGetSetValue

//! Known file types
//@{
#define GS_CLIPMODE_ILLEGAL                         0xFFFFFFFFUL //
filetypeIllegal
#define GS_CLIPMODE_CLIPSPACE               0                   //
filetypeClipSpace
#define GS_CLIPMODE_TCSPACE                         1
        // filetypeTCSpace
#define GS_CLIPMODE_SINGLE                  2                   //
filetypeSingle
#define GS_CLIPMODE_FILM                    3                   //
filetypeFilm
//@}


// Standard values for Get/Set/Supported commands
//! For cmdGetSetValue::gsTcSource - Using LTC
#define GS_TCSOURCE_LTC         1
//! For cmdGetSetValue::gsTcSource - Using VITC
#define GS_TCSOURCE_VITC    2
//! For cmdGetSetValue::gsTcSource - Using CTL
#define GS_TCSOURCE_CTL         4
//! For cmdGetSetValue::gsTcSource - Using absolute clip
#define GS_TCSOURCE_CLIP    7

//! No data, unknown data for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_UNKNOWN    0x00000
//! Ascii data (all printable) for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_ASCII      0x00001
//! Binary (hex) data for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_HEX        0x00002
//! Telecine RP-215 / DPX Data for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_TELECINE   0x10001
//! Close caption/teletext for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_CC_TTEXT   0x10002
//! Navy telemtry data for cmdGetSetValue::gsFrameData,
cmdType::ctSetValue/cmdType::ctGetValue
#define GS_FRAMEDATA_NAVY       0x10003
```

```
//! Set cmdGetSetValue::gsEditMode for an insert edit
#define GS_INSERT_EDIT                  0x01
//! Set cmdGetSetValue::gsEditMode for an assemble edit
#define GS_ASSEMBLE_EDIT    0x02

//! Audio in/out unbalanced (RCA connector) high impedence at -10db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_10              0x001
//! Audio in/out unbalanced (RCA connector) high impedence at -4db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_4               0x002
//! Audio in/out balanced (XLR connector) 600ohm impedence at -10db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_BALANCED_10                0x010
//! Audio in/out balanced (XLR connector) 600ohm impedence at +4db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_BALANCED_4                 0x020
//! Audio in/out digital single wire (cmdGetSetValue::gsAudInSelect
cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_SPDIF                      0x100
//! Audio in/out digital balanced with clock (cmdGetSetValue::gsAudInSelect
cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_AES_EBU                    0x200
//! Audio in/out embedded in SDI or HD-SDI video signal
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_EMBEDDED                   0x400
//! Audio in/out digital balanced with clock (cmdGetSetValue::gsAudInSelect
cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_AES_EBU_PRO                0x800
//! No audio in/out available, or cannot be configured (cmdGetSetValue::gsAudInSelect
cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_NONE                       0
//! No Audio Selected leave silent
#define GS_AUDSELECT_SILENT                             0x040

//Defines for various audio bit rates.
#define GS_AUD_BIT_RATE_32000                   0x0001
#define GS_AUD_BIT_RATE_41100                   0x0002
#define GS_AUD_BIT_RATE_48000                   0x0004
#define GS_AUD_BIT_RATE_56000                   0x0008
#define GS_AUD_BIT_RATE_64000                   0x0010
#define GS_AUD_BIT_RATE_80000                   0x0020
#define GS_AUD_BIT_RATE_96000                   0x0040
#define GS_AUD_BIT_RATE_112000                  0x0080
#define GS_AUD_BIT_RATE_128000                  0x0100
#define GS_AUD_BIT_RATE_160000                  0x0200
#define GS_AUD_BIT_RATE_192000                  0x0400
#define GS_AUD_BIT_RATE_224000                  0x0800
```

```
#define GS_AUD_BIT_RATE_256000                  0x1000
#define GS_AUD_BIT_RATE_320000                  0x2000
#define GS_AUD_BIT_RATE_384000                  0x4000

#define GS_AUD_STEREO                                   0x001
#define GS_AUD_JOINT_STEREO                             0x002
#define GS_AUD_DUAL                                     0x004
#define GS_AUD_SINGLE                                   0x008
#define GS_AUD_MULTIPLE                                 0x010
#define GS_AUD_HEADROOM_18                              0x01
#define GS_AUD_HEADROOM_20                              0x02

//! Freeze - no freeze (cmdGetSetValue::gsVidFreeze)
#define GS_VIDFREEZE_NOT_FROZEN                 0
//! Freeze - first (0) field (cmdGetSetValue::gsVidFreeze)
#define GS_VIDFREEZE_FIELD0                             1
//! Freeze - second (1) field (cmdGetSetValue::gsVidFreeze)
#define GS_VIDFREEZE_FIELD1                             2
//! Freeze - both fields (cmdGetSetValue::gsVidFreeze)
#define GS_VIDFREEZE_FRAME                      3

//! Standard NTSC or PAL composite video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE                          0x001
//! SVHS/S-Video four wire NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SVIDEO                                     0x002
//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE_2                0x004
//! third NTSC or PAL video (often monitor selection) (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE_3                0x008
//! BetaCam level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV                      0x010
//! Panasonic M2 level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_M2           0x020
//! SMPTE standard level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_SMPTE        0x040
//! RGB at video standard rate (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_RGB                      0x080
//! D1 Serial Digital or HDSDI video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_D1_SERIAL                          0x100
```

```
//! D1 Serial Parallel video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_D1_PARALLEL                0x200
//! SDTI/SDI including high speed transfer video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SDTI                       0x400
///Extra fro Digital Rapids--order is screwed up but as long as it works I guess
//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE_4                0x800
//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SVIDEO_2                   0x1000
//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_2            0x2000
//! Secondary NTSC or PAL video (often monitor selection)
(cmdGetSetValue::gsVidInSelect cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_D1_SERIAL_2                0x4000
//! Standard NTSC or PAL composite video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPOSITE_JAPAN            0x8000
//! SVHS/S-Video four wire NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_SVIDEO_JAPAN               0x10000
//! BetaCam level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_JAPAN        0x20000
//! SMPTE standard level YCrCb NTSC or PAL video (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_COMPONENT_YUV_SMPTE_JAPAN  0x40000
//! xVGA compatible analog RGB
#define GS_VIDSELECT_XVID_RGB                   0x80000
//! No video available or no configuratable settings (cmdGetSetValue::gsVidInSelect
cmdGetSetValue::gsVidOutSelect)
#define GS_VIDSELECT_NONE                       0

//! VTR (unruly hsync) lock for cmdGetSetValue::gsVidInLockType
cmdGetSetValue::gsVidOutLockType cmdGetSetValue::gsVidOutLockType
#define GS_VIDLOCKTYPE_VTR                      1
//! Perfect lock for cmdGetSetValue::gsVidInLockType
cmdGetSetValue::gsVidOutLockType cmdGetSetValue::gsVidOutLockType
#define GS_VIDLOCKTYPE_BROADCAST                2

//! Allow normal bandwidth (gsGetSetValue::gsVidInBandwidth
gsGetSetValue::gsVidBandwidth)
#define GS_VIDBAND_STANDARD             0x01
//! Allow medium bandwidth (gsGetSetValue::gsVidInBandwidth
gsGetSetValue::gsVidBandwidth)
```

```
#define GS_VIDBAND_MEDIUM          0x02
//! Allow high bandwidth (gsGetSetValue::gsVidInBandwidth
gsGetSetValue::gsVidBandwidth)
#define GS_VIDBAND_HIGH                    0x04
//! Impose notch filter on bandwidth (gsGetSetValue::gsVidInBandwidth
gsGetSetValue::gsVidBandwidth)
#define GS_VIDBAND_NOTCH           0x08

//! Black at normal level (7.5 IRE NTSC, 0 IRE PAL) gsGetSetValue::gsVidBlackSetup
gsGetSetValue::gsVidInBlack
#define GS_VIDBLACK_SETUP          0x01    // 7.5 ire ntsc, 0 ire pal
//! Crystal black level (0 IRE NTSC, 0 IRE PAL) gsGetSetValue::gsVidBlackSetup
gsGetSetValue::gsVidInBlack
#define GS_VIDBLACK_CRYSTAL                0x02    // 0 ire ntsc, 0 ire pal
//! Super black level (0 > IRE NTSC/PAL) gsGetSetValue::gsVidBlackSetup
gsGetSetValue::gsVidInBlack
#define GS_VIDBLACK_SUPER          0x04    // < 0 ire ntsc/pal

//! Whites are clamped or 100 IRE (gsGetSetValue::gsVidInWhite)
#define GS_VIDWHITE_CLAMP          0x01    // 100 IRE white max
//! Whites are scaled automatically from black level to 100 IRE
(gsGetSetValue::gsVidInWhite)
#define GS_VIDWHITE_SCALE          0x02    // Scale like AGC
//! Whites are allowed to be greater then 100 IRE (gsGetSetValue::gsVidInWhite)
#define GS_VIDWHITE_FREE           0x04    // Any white

//! No external genlock source (free running on internal clock)
(gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_NONE            0x0001 // No genlock (free run master)
//! External ref in is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_EXTIN   0x0002 // Lock to external in
//! Current input is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_INPUT   0x0004 // Lock to current input
//! Composite (CVBS) input is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_CVBS            0x0008 // Lock to composite ing
//! S-Video (SVHS) input is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_SVIDEO  0x0010 // Lock to S-Video In
//! Component Y input is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_IN_Y            0x0020 // Lock to Y of betacam input
//! SDI serial digital input is genlock source (gsGetSetValue::gsVidOutGenlockSource)
#define GS_LOCKSRC_SDI             0x0040 // Lock to Digital Module Ref
In. //DigiSuiteLite

//! Keep analog monitor in line with digital (hd=hd, sd=sd)
#define GS_ANALOGMONITORMETHOD_DIRECT      0x0000
//! Convert everything to the nearest SD type
#define GS_ANALOGMONITORMETHOD_SD          0x0001
//! Convert everything to the nearest 720 HD type
#define GS_ANALOGMONITORMETHOD_HD7200x0002
```

```
//! Convert everything to the nearest 1080 HD type
#define GS_ANALOGMONITORMETHOD_HD1080       0x0003
//! SD->HD720 and HD->SD
#define GS_ANALOGMONITORMETHOD_FLIP720      0x0004
//! SD->HD720 and HD->SD
#define GS_ANALOGMONITORMETHOD_FLIP1080     0x0005

//! Upconvert to whole screen
#define GS_UPCONVERT_ANAMORPHIC                 0x0001
//! Upconvert with bars
#define GS_UPCONVERT_PILLARBOX              0x0002
//! Upconvert with some zoom
#define GS_UPCONVERT_ZOOM14x9               0x0004
//! Upconvert to letter box
#define GS_UPCONVERT_LETTERBOX              0x0008
//! Upconvert to wide zoom
#define GS_UPCONVERT_ZOOMWIDE               0x0010

//! Down convert with top/bottom black bars
#define GS_DOWNCONVERT_LETTERBOX            0x0001
//! Down convert and crop image
#define GS_DOWNCONVERT_CROP                     0x0002
//! Down convert to whole screen
#define GS_DOWNCONVERT_ANAMORPHIC           0x0004

//! Standard MPEG resolution 120
#define GS_MPEG_RESOLUTION_120              0x001
//! Standard MPEG resolution 240
#define GS_MPEG_RESOLUTION_240              0x002
//! Standard MPEG resolution 288
#define GS_MPEG_RESOLUTION_288              0x004
//! Standard MPEG resolution 352
#define GS_MPEG_RESOLUTION_352              0x008
//! Standard MPEG resolution 480
#define GS_MPEG_RESOLUTION_480              0x010
//! Standard MPEG resolution 512
#define GS_MPEG_RESOLUTION_512              0x020
//! Standard MPEG resolution 522
#define GS_MPEG_RESOLUTION_544              0x040
//! Standard MPEG resolution 576
#define GS_MPEG_RESOLUTION_576              0x080
//! Standard MPEG resolution 608
#define GS_MPEG_RESOLUTION_608              0x100
//! Standard MPEG resolution 704
#define GS_MPEG_RESOLUTION_704              0x200
//! Standard MPEG resolution 720
#define GS_MPEG_RESOLUTION_720              0x400

//! Chroma format 4:2:0
```

```
#define GS_CHROMA_FORMAT_420                    0x1
//! Chroma format 4:2:2
#define GS_CHROMA_FORMAT_422                    0x2
//! Chroma format 4:4:4
#define GS_CHROMA_FORMAT_444                    0x4
//! Chroma format 4:1:1
#define GS_CHROMA_FORMAT_411                    0x8

//! MPEG chroma format 4:2:0 see #GS_CHROMA_FORMAT_420
#define GS_MPEG_CHROMA_FORMAT_420               GS_CHROMA_FORMAT_420
//! MPEG chroma format 4:2:2 see #GS_CHROMA_FORMAT_422
#define GS_MPEG_CHROMA_FORMAT_422               GS_CHROMA_FORMAT_422
//! MPEG chroma format 4:4:4 see #GS_CHROMA_FORMAT_444
#define GS_MPEG_CHROMA_FORMAT_444               GS_CHROMA_FORMAT_444

//! MPEG DCT Precision 8 bits
#define GS_MPEG_DC_PRECISION_8                  0x1
//! MPEG DCT Precision 9 bits
#define GS_MPEG_DC_PRECISION_9                  0x2
//! MPEG DCT Precision 10 bits
#define GS_MPEG_DC_PRECISION_10                 0x4
//! MPEG DCT Precision 11 bits
#define GS_MPEG_DC_PRECISION_11                 0x8

//! Aspect ratio square
#define GS_ASPECT_RATIO_SQUARE          0x1
//! Aspect ratio 4:3
#define GS_ASPECT_RATIO_4x3                     0x2
//! Aspect ratio 16:9
#define GS_ASPECT_RATIO_16x9            0x4
//! Aspect ratio 2.21:1
#define GS_ASPECT_RATIO_2_21x1          0x8
//! MPEG aspect ratio square see #GS_ASPECT_RATIO_SQUARE
#define GS_MPEG_ASPECT_RATIO_SQUARE             GS_ASPECT_RATIO_SQUARE
//! MPEG aspect ratio square see #GS_ASPECT_RATIO_4x3
#define GS_MPEG_ASPECT_RATIO_4x3        GS_ASPECT_RATIO_4x3
//! MPEG aspect ratio square see #GS_ASPECT_RATIO_16x9
#define GS_MPEG_ASPECT_RATIO_16x9               GS_ASPECT_RATIO_16x9
//! MPEG aspect ratio square see #GS_ASPECT_RATIO_2_21x1
#define GS_MPEG_ASPECT_RATIO_2_21x1             GS_ASPECT_RATIO_2_21x1

#define GS_MPEG_STANDARD_SYSTEM                         0x1
#define GS_MPEG_STANDARD_PROGRAM                0x2
#define GS_MPEG_STANDARD_TRANSPORT              0x4
#define GS_MPEG_STANDARD_ELEMENTARY             0x8
#define GS_MPEG_STANDARD_ELEMENTRY
        GS_MPEG_STANDARD_ELEMENTARY

#define GS_MPEG_LANGAUGE_ENGLISH                0x0001
```

```
#define GS_MPEG_LANGAUGE_SPANISH                0x0002
#define GS_MPEG_LANGAUGE_FRENCH                         0x0004
#define GS_MPEG_LANGAUGE_GERMAN                         0x0008
#define GS_MPEG_LANGAUGE_JAPANESE               0x0010
#define GS_MPEG_LANGAUGE_DUTCH                  0x0020
#define GS_MPEG_LANGAUGE_DANISH                 0x0040
#define GS_MPEG_LANGAUGE_FINNISH                0x0080
#define GS_MPEG_LANGAUGE_ITALIAN                0x0100
#define GS_MPEG_LANGAUGE_GREEK                  0x0200
#define GS_MPEG_LANGAUGE_PORTUGUESE             0x0400
#define GS_MPEG_LANGAUGE_SWEDISH                0x0800
#define GS_MPEG_LANGAUGE_RUSSIAN                0x1000
#define GS_MPEG_LANGAUGE_CHINESE                0x2000

#define GS_MPEG_CC_FORMAT_CCUBE                         0x1
#define GS_MPEG_CC_FORMAT_ATSC                  0x2
#define GS_MPEG_CC_FORMAT_CCUBE_REORDER 0x4
#define GS_MPEG_CC_FORMAT_ATSC_REORDER          0x8

#define GS_MPEG_ONE_FRAMES                              0x0001
#define GS_MPEG_TWO_FRAMES                              0x0002
#define GS_MPEG_THREE_FRAMES                    0x0004
#define GS_MPEG_FOUR_FRAMES                             0x0008
#define GS_MPEG_FIVE_FRAMES                             0x0010
#define GS_MPEG_SIX_FRAMES                              0x0020
#define GS_MPEG_SEVEN_FRAMES                    0x0040
#define GS_MPEG_EIGHT_FRAMES                    0x0080
#define GS_MPEG_NINE_FRAMES                             0x0100
#define GS_MPEG_TEN_FRAMES                              0x0200
#define GS_MPEG_ELEVEN_FRAMES                   0x0400
#define GS_MPEG_TWELVE_FRAMES                   0x0800
#define GS_MPEG_THIRTEEN_FRAMES                 0x1000
#define GS_MPEG_FOURTEEN_FRAMES                 0x2000
#define GS_MPEG_FIFTEEN_FRAMES                  0x4000
#define GS_MPEG_SIXTEEN_FRAMES                  0x8000

//! Video for windows avi (audio video interleave)
#define VIDEOWRITETYPE_AVI          0x00000001      //standard uncompressed AVI
//! QuickTime movie (apple)
#define VIDEOWRITETYPE_MOV                  0x00000002      //Uncompressed
Quicktime
//! Windows Media Video (microsoft)
#define VIDEOWRITETYPE_WMV                  0x00000004      //uncompressed
windows media
//! SoftImage/Avid uncompressed GEN
#define VIDEOWRITETYPE_GEN                  0x00000008      //Avid uncompressed
Gen File
//! Jaleo uncompressed format
#define VIDEOWRITETYPE_JS           0x00000010      //Uncompressed Jaleo
```

```
//! DVS file format
#define VIDEOWRITETYPE_DVS                  0x00000020      //
//! Iridas 8 bit rgb format
#define VIDEOWRITETYPE_IHSS                 0x00000040      //
//! HDR+Raw descriptor for raw streams
#define VIDEOWRITETYPE_HDR                  0x00000080      //Header YUV
//! Stills - 8/10 bit YCbCr .yuv or .v210
#define VIDEOWRITETYPE_YUV                  0x00000100      //YUV still file format
//! Stills - Raw 24/32 bit RGB/RGBA
#define VIDEOWRITETYPE_RAW                  0x00000200      //uncompressed raw still
file format
//! Stills - Targa 24/32 bit RGB
#define VIDEOWRITETYPE_TGA                  0x00000400      //Targa still file format
//! Stills - no longer supported
#define VIDEOWRITETYPE_BMP                  0x00000800      //birmap still file format
//! Stills - Tiff 24/32 bit RGB/RGBA
#define VIDEOWRITETYPE_TIFF                 0x00001000      //TIFF still file format
//! Stills - Cineon 10 bit RGB
#define VIDEOWRITETYPE_CIN          0x00002000      //cineon still file format
//! Stills - DPX (Smpte/Kodak) 10 bit RGB
#define VIDEOWRITETYPE_DPX                  0x00004000      //dpx still file format
//! MPEG stream
#define VIDEOWRITETYPE_MPG                  0x00008000      //mpeg
//! Stills - 4224 individual frames of 8 or 10 bit YCbCr+A
#define VIDEOWRITETYPE_4224                 0x00010000      //4:2:2:4 YCbCr 8/10 bit
//! MXF - compress or uncompressed streams
#define VIDEOWRITETYPE_MXF                  0x00020000      //mxf D10/HDV

//! Turn off monitor
#define GS_MONITORGRAB_NONE                         0x0000
//! Type mask (jpg, bmp)
#define GS_MONITORGRAB_TYPE_MASK        0x00F0
//! Use BMP format for image
#define GS_MONITORGRAB_TYPE_BMP         0x0000
//! Use JPEG format for image
#define GS_MONITORGRAB_TYPE_JPG         0x0010
//! Size mask (full, half, quarter)
#define GS_MONITORGRAB_SIZE_MASK        0x000F
//! Full size image captured
#define GS_MONITORGRAB_SIZE_FULL        0x0001
//! Half size image captured
#define GS_MONITORGRAB_SIZE_HALF        0x0002
//! Quarter size image captured
#define GS_MONITORGRAB_SIZE_QUARTER     0x0004
//! Target/To mask
#define GS_MONITORGRAB_TARGET_MASK      0x0F00
//! Use the arbID area
#define GS_MONITORGRAB_TO_MEMORY        0x0100
//! Save image to a UNC path
```

```
#define GS_MONITORGRAB_TO_UNC_PATH     0x0200
//! Save image to web server (use name sent in arbID)
#define GS_MONITORGRAB_TO_HTTP         0x0400
//! Save image through 'to be announced' network transport
#define GS_MONITORGRAB_TO_NETWORK      0x0800

//@{
#define GS_USERRIGHTS_NONE             0x0000
#define GS_USERRIGHTS_READ             0x0001
#define GS_USERRIGHTS_MODIFY       0x0002
#define GS_USERRIGHTS_WRITE            0x0004
#define GS_USERRIGHTS_SETUP            0x0008
#define GS_USERRIGHTS_PLAY         0x0010
#define GS_USERRIGHTS_RECORD       0x0020
#define GS_USERRIGHTS_ADD          0x0100
#define GS_USERRIGHTS_DELETE       0x0200
#define GS_USERRIGHTS_FULL         0x7FFF
#define GS_USERRIGHTS_ADMIN            0x8000
//@}

//
// Masks and shifts
//
//! Frame rate mask (portion of return for frame rate)
#define GS_SIGFORMMASK_FRAMERATE       0x000001ff
//! Shift frame rate to 0
#define GS_SIGFORMSHIFT_FRAMERATE      0
//! Horizontal / 8 mask (portion of return for frame rate)
#define GS_SIGFORMMASK_HORIZONTAL      0x000ffe00
//! Horizontal / 8 shift to 0
#define GS_SIGFORMSHIFT_HORIZONTAL     9
//! Vertical / 8 mask (portion of return for frame rate)
#define GS_SIGFORMMASK_VERTICAL        0x0ff00000
//! Vertical / 8 shift to 0
#define GS_SIGFORMSHIFT_VERTICAL 20
//! Frame type mask (portion of return for frame rate)
#define GS_SIGFORMMASK_FRAMETYPE       0xF0000000UL
//! Frame type shift to 0
#define GS_SIGFORMSHIFT_FRAMETYPE      28

//
// Basic frame rate types
//
#define GS_SIGFORMFRAMERATE_5          5
#define GS_SIGFORMFRAMERATE_6          6
#define GS_SIGFORMFRAMERATE_7_5        7
#define GS_SIGFORMFRAMERATE_10         10
#define GS_SIGFORMFRAMERATE_14_98      14
#define GS_SIGFORMFRAMERATE_15         15
```

```
#define GS_SIGFORMFRAMERATE_23_98       23
#define GS_SIGFORMFRAMERATE_24          24
#define GS_SIGFORMFRAMERATE_25          25
#define GS_SIGFORMFRAMERATE_29_97       29
#define GS_SIGFORMFRAMERATE_30          30
#define GS_SIGFORMFRAMERATE_47_95       47
#define GS_SIGFORMFRAMERATE_48          48
#define GS_SIGFORMFRAMERATE_50          50
#define GS_SIGFORMFRAMERATE_59_94       59
#define GS_SIGFORMFRAMERATE_60          60
#define GS_SIGFORMFRAMERATE_71_93       71
#define GS_SIGFORMFRAMERATE_72          72
#define GS_SIGFORMFRAMERATE_100         100             // 0x64
#define GS_SIGFORMFRAMERATE_119_88      119             // 0x77
#define GS_SIGFORMFRAMERATE_CUSTOM      0x100

//
// Sizing elements
//
#define GS_SIGFORMSIZE_240              0x01
#define GS_SIGFORMSIZE_243              0x02
#define GS_SIGFORMSIZE_288              0x03    // Gap 4
#define GS_SIGFORMSIZE_320              0x08
#define GS_SIGFORMSIZE_352              0x09
#define GS_SIGFORMSIZE_360              0x0a    // Gap 5
#define GS_SIGFORMSIZE_480              0x10
#define GS_SIGFORMSIZE_483              0x11
#define GS_SIGFORMSIZE_486              0x12    // Gap 1
#define GS_SIGFORMSIZE_496              0x14    // Gap 2
#define GS_SIGFORMSIZE_504              0x16
#define GS_SIGFORMSIZE_512              0x17    // Gap 2
#define GS_SIGFORMSIZE_576              0x1a
#define GS_SIGFORMSIZE_600              0x1b
#define GS_SIGFORMSIZE_608              0x1c    // Gap 3
#define GS_SIGFORMSIZE_640              0x20
#define GS_SIGFORMSIZE_720              0x21
#define GS_SIGFORMSIZE_768              0x22
#define GS_SIGFORMSIZE_800              0x23
#define GS_SIGFORMSIZE_864              0x24
#define GS_SIGFORMSIZE_988              0x25    // Gap 3
#define GS_SIGFORMSIZE_857              0x26
#define GS_SIGFORMSIZE_960              0x28
#define GS_SIGFORMSIZE_968              0x29
#define GS_SIGFORMSIZE_778              0x2A
#define GS_SIGFORMSIZE_872              0x2B    // Gap 4
#define GS_SIGFORMSIZE_1024             0x30
#define GS_SIGFORMSIZE_1035             0x31
#define GS_SIGFORMSIZE_1044             0x32
#define GS_SIGFORMSIZE_1052             0x33
```

```
#define GS_SIGFORMSIZE_1050                     0x34    // Gap 4
#define GS_SIGFORMSIZE_1080                     0x38
#define GS_SIGFORMSIZE_1088                     0x39
#define GS_SIGFORMSIZE_1096                     0x3a    // Gap 3
#define GS_SIGFORMSIZE_1102                     0x3E    // Gap 1
#define GS_SIGFORMSIZE_1152                     0x40
#define GS_SIGFORMSIZE_1200                     0x41    // Gap 1
#define GS_SIGFORMSIZE_1234                     0x43    // Gap 6
#define GS_SIGFORMSIZE_1280                     0x48
#define GS_SIGFORMSIZE_1332                     0x49    // Gap 1
#define GS_SIGFORMSIZE_1400                     0x4B
#define GS_SIGFORMSIZE_1440                     0x4C    // Gap 3
#define GS_SIGFORMSIZE_1536                     0x50
#define GS_SIGFORMSIZE_1556                     0x51
#define GS_SIGFORMSIZE_1588                     0x52    // Gap 4
#define GS_SIGFORMSIZE_1828                     0x56
#define GS_SIGFORMSIZE_1714                     0x57
#define GS_SIGFORMSIZE_1600                     0x58
#define GS_SIGFORMSIZE_1920                     0x59    // Gap 7
#define GS_SIGFORMSIZE_2048                     0x60    // Gap 8
#define GS_SIGFORMSIZE_2650                     0x68    // Gap 4
#define GS_SIGFORMSIZE_3112                     0x6b    // Gap 3
#define GS_SIGFORMSIZE_4096                     0x80

//
// Basic Frame Sizes
//
#define GS_SIGFORMSIZE_640x480          ((GS_SIGFORMSIZE_640 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_480 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_640x576          ((GS_SIGFORMSIZE_640 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_576 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x480          ((GS_SIGFORMSIZE_720 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_480 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x483          ((GS_SIGFORMSIZE_720 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_483 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x486          ((GS_SIGFORMSIZE_720 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_486 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x512          ((GS_SIGFORMSIZE_720 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_512 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x576          ((GS_SIGFORMSIZE_720 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_576 <<
GS_SIGFORMSHIFT_VERTICAL))
```

```
#define GS_SIGFORMSIZE_720x608          ((GS_SIGFORMSIZE_720 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_608 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_720x504          ((GS_SIGFORMSIZE_720 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_504 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_800x600          ((GS_SIGFORMSIZE_800 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_600 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_960x486          ((GS_SIGFORMSIZE_960 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_486 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_960x576          ((GS_SIGFORMSIZE_960 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_576 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_960x504          ((GS_SIGFORMSIZE_960 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_504 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1024x768         ((GS_SIGFORMSIZE_1024 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_768 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1024x1024  ((GS_SIGFORMSIZE_1024 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1024 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1152x864         ((GS_SIGFORMSIZE_1152 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_864 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1280x1024  ((GS_SIGFORMSIZE_1280 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1024 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1400x1050  ((GS_SIGFORMSIZE_1400 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1050 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1600x1200  ((GS_SIGFORMSIZE_1600 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1200 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1280x720         ((GS_SIGFORMSIZE_1280 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_720 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1828x778     ((GS_SIGFORMSIZE_1828 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_778 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1828x988     ((GS_SIGFORMSIZE_1828 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_988 << GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1828x1102    ((GS_SIGFORMSIZE_1828 << GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1102 << GS_SIGFORMSHIFT_VERTICAL))
```

```
#define GS_SIGFORMSIZE_1828x1332    ((GS_SIGFORMSIZE_1828 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1332 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1920x1035  ((GS_SIGFORMSIZE_1920 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1035 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1920x1080  ((GS_SIGFORMSIZE_1920 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1080 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_1920x1088  ((GS_SIGFORMSIZE_1920 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1088 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2560x1080  ((GS_SIGFORMSIZE_2650 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1080 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x857     ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_857 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x872     ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_872 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x1102    ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1102 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x1234    ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1234 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x1080  ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1080 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x1536  ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1536 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_2048x1556  ((GS_SIGFORMSIZE_2048 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1556 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_4096x1714  ((GS_SIGFORMSIZE_4096 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_1714 <<
GS_SIGFORMSHIFT_VERTICAL))
#define GS_SIGFORMSIZE_4096x3112  ((GS_SIGFORMSIZE_4096 <<
GS_SIGFORMSHIFT_HORIZONTAL) | (GS_SIGFORMSIZE_3112 <<
GS_SIGFORMSHIFT_VERTICAL))

//
// Basic frame types
//
#define GS_SIGFORMTYPE_UNKNOWN                  (0)
#define GS_SIGFORMTYPE_INTERLACED               (1 <<
GS_SIGFORMSHIFT_FRAMETYPE)
```

```
#define GS_SIGFORMTYPE_PROGRESSIVE              (2 <<
GS_SIGFORMSHIFT_FRAMETYPE)
#define GS_SIGFORMTYPE_SEGMENTEDFRAME(4 << GS_SIGFORMSHIFT_FRAMETYPE)

//! Signal format NTSC square pixel (320x240 or 640x480) @ 29.97 or 30 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_NTSC                         (GS_SIGFORMSIZE_640x480 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)
//! Signal format PAL square pixel (320x288 or 640x576) @ 25 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_PAL                          (GS_SIGFORMSIZE_640x576 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_25)
//! Signal format NTSC square pixel (360/352x243/240 or 720/704x486/480) @ 29.97 or
30 fps gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CCIR_NTSC            (GS_SIGFORMSIZE_720x486 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)
//! Signal format NTSC square pixel (360/352x243/240 or 720/704x486/480) @ 29.97 or
30 fps gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CCIR_NTSC_P483               (GS_SIGFORMSIZE_720x483 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_29_97)
//! Signal format PAL square pixel (360/352x288 or 720/704x576) @ 25 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CCIR_PAL                     (GS_SIGFORMSIZE_720x576 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_25)
//! Signal format NTSC at 30 hz Progressive
#define GS_SIGFORM_CCIR_PNTSC_30(GS_SIGFORMSIZE_720x486 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_30)
//! Signal format PAL at 25 hz Progressive
#define GS_SIGFORM_CCIR_PPAL_25         (GS_SIGFORMSIZE_720x576 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)
//! Signal format compressed HD 960x504 29.97
#define GS_SIGFORM_HD360                        (GS_SIGFORMSIZE_960x504 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)
//! Signal format NTSC High Res (960x486)
#define GS_SIGFORM_ALT_NTSC                     (GS_SIGFORMSIZE_960x486 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)
//! Signal format PAL High Res (960x576)
#define GS_SIGFORM_ALT_PAL              (GS_SIGFORMSIZE_960x576 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_25)

//! 2200x1125 raster, 1920x1035 production aperture (1888x1017 clean) @ 30 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1035i_30_260M (GS_SIGFORMSIZE_1920x1035 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_30)
//! 2200x1125 raster, 1920x1035 production aperture (1888x1017 clean) @ 29.97 fp
gsGetSetValue::gsSignalFormats
#define GS_SIGFORM_1035i_30X_260M       (GS_SIGFORMSIZE_1920x1035 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)
//! 1920x1080i (274M-1997 Table1 System 4) @ 29.97 gsGetSetValue::gsSignalFormat
```

```
#define GS_SIGFORM_1080i_30                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_30)        /*
        (274M-1997 Table1 System 4)  */
#define GS_SIGFORM_1080sf_30                   (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_30)         /*
        (274M-1997 Table1 System 4)  */
//! 1920x1080i (274M-1997 Table1 System 4) @ 30 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_30X                   (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_29_97)    /*
        (274M-1997 Table1 System 5)  */
#define GS_SIGFORM_1080sf_30X                  (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_29_97)     /*
        (274M-1997 Table1 System 5)  */
//! 1920x1080i (274M-1997 Table1 System 4) @ 25 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_25                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_25)        /*
        (274M-1997 Table1 System 6)  */
#define GS_SIGFORM_1080sf_25                   (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_25)         /*
        (274M-1997 Table1 System 6)  */
//! 1920x1080sf (274M-1997 Table1 System 4) @ 24 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_24                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_24)
#define GS_SIGFORM_1080sf_24                   (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_24)
//! 1920x1080sf (274M-1997 Table1 System 4) @ 23.98 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_24X                   (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_23_98)
#define GS_SIGFORM_1080sf_24X                  (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_23_98)
//! 1920x1080P (274M-1997 Table1 System 4) @ 30 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_30                     (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_30)       /*
        (274M-1997 Table1 System 7)  */
//! 1920x1080P (274M-1997 Table1 System 4) @ 29.97 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_30X                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_29_97)  /*
        (274M-1997 Table1 System 8)  */
//! 1920x1080P (274M-1997 Table1 System 4) @ 25 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_25                     (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)       /*
        (274M-1997 Table1 System 9)  */
//! 1920x1080P (274M-1997 Table1 System 4) @ 24 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_24                     (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)       /*
        (274M-1997 Table1 System 10) */
//! 1920x1080P (274M-1997 Table1 System 4) @ 23.98 gsGetSetValue::gsSignalFormat
```

```
#define GS_SIGFORM_1080_24X                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)  /*
        (274M-1997 Table1 System 11) */


//! 1920x1080P 60 (Dual P30)
#define GS_SIGFORM_1080_60             (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_60)
//! 1920x1080P 59.94 (Dual P29.97)
#define GS_SIGFORM_1080_60X                    (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_59_94)
//! 1920x1080P 50 (Dual 25)
#define GS_SIGFORM_1080_50             (GS_SIGFORMSIZE_1920x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_50)


//! 1650x750 raster, 1280x720 production aperture (1248x702 clean): @ 60
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_720_60              (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_60)       /*
        (296M-1996 Table1 System 1)   */
//! 1650x750 raster, 1280x720 production aperture (1248x702 clean): @ 59.97
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_720_60X             (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_59_94)  /*
        (296M-1996 Table1 System 2)   */
//! 50 Hz DVS, IRT
#define GS_SIGFORM_720_50              (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_50)       /*
        (296M-1996 Table1 System 1)   */

//! Half frame rate 720/60
#define GS_SIGFORM_720_30              (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_30)
//! Half frame rate 720/59.94
#define GS_SIGFORM_720_30X             (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_29_97)
//! Half 50 Hz DVS, IRT
#define GS_SIGFORM_720_25              (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)       /*
        (296M-1996 Table1 System 1)   */
//! 720x1280 true 24 (varicam)
#define GS_SIGFORM_720_24              (GS_SIGFORMSIZE_1280x720 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)       /*
        (296M-1996 Table1 System 1)   */

/** VGA res
*/
//! gsGetSetValue::gsSignalFormat Vesa 640x480@72
```

```
#define GS_SIGFORM_VESA_640_72          (GS_SIGFORMSIZE_640x480 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_72)
//! gsGetSetValue::gsSignalFormat Vesa 800x600@71.9
#define GS_SIGFORM_VESA_800_71X         (GS_SIGFORMSIZE_800x600 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_71_93)
//! gsGetSetValue::gsSignalFormat Vesa 800x600@72
#define GS_SIGFORM_VESA_800_72          (GS_SIGFORMSIZE_800x600 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_72)
//! gsGetSetValue::gsSignalFormat Vesa 1024x768@71.9
#define GS_SIGFORM_VESA_1024_71X (GS_SIGFORMSIZE_1024x768 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_71_93)
//! gsGetSetValue::gsSignalFormat Vesa 1024x766@72
#define GS_SIGFORM_VESA_1024_72         (GS_SIGFORMSIZE_1024x768 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_72)
//! gsGetSetValue::gsSignalFormat Vesa 1280x1024@24
#define GS_SIGFORM_VESA_1280_24         (GS_SIGFORMSIZE_1280x1024 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Vesa 1280x1024@30
#define GS_SIGFORM_VESA_1280i_30  (GS_SIGFORMSIZE_1280x1024 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_30)
//! gsGetSetValue::gsSignalFormat Vesa 1280x1024@71.9
#define GS_SIGFORM_VESA_1280_71X (GS_SIGFORMSIZE_1280x1024 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_71_93)
//! gsGetSetValue::gsSignalFormat Vesa 1280x1024@72
#define GS_SIGFORM_VESA_1280_72         (GS_SIGFORMSIZE_1280x1024 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_72)
//! gsGetSetValue::gsSignalFormat Vesa 1600x1200i@30
#define GS_SIGFORM_VESA_1600i_30  (GS_SIGFORMSIZE_1600x1200 |
GS_SIGFORMTYPE_INTERLACED | GS_SIGFORMFRAMERATE_30)

/** Presentation res
*/
//! gsGetSetValue::gsSignalFormat Presentation
#define GS_SIGFORM_DVI_1400_1050_24     (GS_SIGFORMSIZE_1400x1050 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Presentation
#define GS_SIGFORM_DVI_1400_1050_25     (GS_SIGFORMSIZE_1400x1050 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Presentation
#define GS_SIGFORM_DCIN_2048_25         (GS_SIGFORMSIZE_2048x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)

/** Digital cinema 2048x1080
*/
//! gsGetSetValue::gsSignalFormat Digtal Cinema
#define GS_SIGFORM_DCIN_2048sf_24X      (GS_SIGFORMSIZE_2048x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Digtal Cinema
```

```
#define GS_SIGFORM_DCIN_2048sf_24          (GS_SIGFORMSIZE_2048x1080 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Digtal Cinema
#define GS_SIGFORM_DCIN_2048_24X (GS_SIGFORMSIZE_2048x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Digtal Cinema
#define GS_SIGFORM_DCIN_2048_24          (GS_SIGFORMSIZE_2048x1080 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)

//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_778_24  (GS_SIGFORMSIZE_1828x778 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_778_25  (GS_SIGFORMSIZE_1828x778 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_988_24  (GS_SIGFORMSIZE_1828x988 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_988_25  (GS_SIGFORMSIZE_1828x988 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_1102_24 (GS_SIGFORMSIZE_1828x1102 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_1102_25 (GS_SIGFORMSIZE_1828x1102 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_1332_24 (GS_SIGFORMSIZE_1828x1332 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_1828_1332_25 (GS_SIGFORMSIZE_1828x1332 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)

//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_857_24  (GS_SIGFORMSIZE_2048x857 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_857_25  (GS_SIGFORMSIZE_2048x857 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)

//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_872_24  (GS_SIGFORMSIZE_2048x872 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_872_25  (GS_SIGFORMSIZE_2048x872 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film
```

```
#define GS_SIGFORM_FILM_2048_1102_24 (GS_SIGFORMSIZE_2048x1102 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_1102_25 (GS_SIGFORMSIZE_2048x1102 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_1234_24 (GS_SIGFORMSIZE_2048x1234 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film
#define GS_SIGFORM_FILM_2048_1234_25 (GS_SIGFORMSIZE_2048x1234 Or
GS_SIGFORMTYPE_PROGRESSIVE Or GS_SIGFORMFRAMERATE_25)

/** Film transfer
*/
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_15X (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_14_98)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_14          GS_SIGFORM_FILM_2048_15X  //
Deprecated, do not use
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_15          (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_15)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048sf_15X       (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_14_98)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048sf_15        (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_15)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_24X (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_24          (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048sf_24X       (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048sf_24(GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 2K
#define GS_SIGFORM_FILM_2048_48          (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_48)

//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_25          (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
```

```
#define GS_SIGFORM_FILM_2048_1536sf_25          (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_25
        (GS_SIGFORMSIZE_2048x1556 | GS_SIGFORMTYPE_PROGRESSIVE |
GS_SIGFORMFRAMERATE_25)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048sf_25               (GS_SIGFORMSIZE_2048x1556 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_25)

//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_15X           (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_14_98)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_15            (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_15)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536sf_15X         (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_14_98)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536sf_15          (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_15)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_24X           (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_24            (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536sf_24X         (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_23_98)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536sf_24          (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_48X           (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_47_95)
//! gsGetSetValue::gsSignalFormat Film 2K(1536)
#define GS_SIGFORM_FILM_2048_1536_48            (GS_SIGFORMSIZE_2048x1536 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_48)

//! gsGetSetValue::gsSignalFormat Film 4K Half
#define GS_SIGFORM_FILM_4096_1714_24            (GS_SIGFORMSIZE_4096x1714 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 4K Half
#define GS_SIGFORM_FILM_4096_1714_24X           (GS_SIGFORMSIZE_4096x1714 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)

//! gsGetSetValue::gsSignalFormat Film 4K
```

```
#define GS_SIGFORM_FILM_4096_3112sf_5          (GS_SIGFORMSIZE_4096x3112 |
GS_SIGFORMTYPE_SEGMENTEDFRAME | GS_SIGFORMFRAMERATE_5)
//! gsGetSetValue::gsSignalFormat Film 4K
#define GS_SIGFORM_FILM_4096_3112_24           (GS_SIGFORMSIZE_4096x3112 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_24)
//! gsGetSetValue::gsSignalFormat Film 4K
#define GS_SIGFORM_FILM_4096_3112_24X          (GS_SIGFORMSIZE_4096x3112 |
GS_SIGFORMTYPE_PROGRESSIVE | GS_SIGFORMFRAMERATE_23_98)


//! All non video rate types (eg. 15fps, 10fps, 37fps) gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CUSTOM                  0xF0000000UL
//! For input and genlock status returns
#define GS_SIGFORM_NOT_PRESENT             0

/**
* Supported formats
*/
//! NTSC (CCIR or sqp) 720x480/486/508/512
#define GS_SIGFORM_SUPPORTS_NTSC           0x00000001
//! PAL (CCIR or sqp) 720x576/608
#define GS_SIGFORM_SUPPORTS_PAL            0x00000002
//! 960 width SD, not used
#define GS_SIGFORM_SUPPORTS_HR             0x00000004
//! 360 compressed, not used
#define GS_SIGFORM_SUPPORTS_360            0x00000008
//! 720p Rasters (59/60 and sometimes 50)
#define GS_SIGFORM_SUPPORTS_720            0x00000010
//! 1035x1080 Production rasters
#define GS_SIGFORM_SUPPORTS_1035           0x00000020
//! 1080/1088/1092/1112x1920 HD rasters
#define GS_SIGFORM_SUPPORTS_1080           0x00000040
//! 1088 HD rasters
#define GS_SIGFORM_SUPPORTS_EXTRA8         0x00000080          // 728, 1044,
1088
//! Film 2K 1536 lines
#define GS_SIGFORM_SUPPORTS_1536           0x00000100          // 2048x1536
//! Film 2K 1556 lines
#define GS_SIGFORM_SUPPORTS_1556           0x00000200          // 2048x1556
//! Digital Cinema 2048x1080
#define GS_SIGFORM_SUPPORTS_DCIN           0x00000400          // 2048x1080
//! Presentation 1440x1050
#define GS_SIGFORM_SUPPORTS_1400           0x00000800          // 1440x1050
//! Vesa 640x480
#define GS_SIGFORM_SUPPORTS_V480           0x00010000          // Vesa 640x480
//! Vesa 800x600
#define GS_SIGFORM_SUPPORTS_V600           0x00020000          // Vesa 800x600
//! Vesa 1024x768
```

```
#define GS_SIGFORM_SUPPORTS_V768        0x00040000              // Vesa
1024x768
//! Vesa 1280x1024
#define GS_SIGFORM_SUPPORTS_V1024       0x00080000              // Vesa
1280x1024
//! Vesa 1600x1200
#define GS_SIGFORM_SUPPORTS_V1200       0x00100000              // Vesa
1600x1200
//! Vesa 1600
#define GS_SIGFORM_SUPPORTS_V1600       0x00200000
//! Modifier times 2
#define GS_SIGFORM_SUPPORTS_X2          0x20000000
//! Modifier times 3
#define GS_SIGFORM_SUPPORTS_X3          0x40000000
//! Modifier times 4
#define GS_SIGFORM_SUPPORTS_X4          0x80000000


//! Software passed codec on main processor (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_SOFTWARE    0x00000001
//! Motion JPEG hardware codec (LSI, Zoran, C-Cube, etc)
(gsGetSetValue::gsCompType)
#define GS_COMPTYPE_MJPEG       0x00000002
//! MPEG-4 h.264
#define GS_COMPTYPE_H264        0x00000004
//! Wavelet hardware codec (gsGetSetValue::gsCompType)
////#define GS_COMPTYPE_WAVELET         0x00000008
#define GS_COMPTYPE_JPEG2000    0x00000008
//! MPEG 1 hardware compatible codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_MPEG1       0x00000010
//! MPEG 2 hardware compatible codec (gsGetSetValue::gsCompType)
/////#define GS_COMPTYPE_MPEG2           0x00000020
//! Editable MPEG 2 I Frame Only compatible codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_MPEG2I              0x00000040
//! MPEG 2 long gop hardware compatible codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_MPEG2IBP    0x00000080
//! Hardware DV25, DVCPRO. DVCPRO25  (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_DV25        0x00000100          // DV25, DVCPRO.
DVCPRO25
//! Hardware DV50, DVCPRO50 (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_DV50        0x00000200          // DV50, DVCPRO50
//! Hardware Standard DV Bluebook, DVPRO, DVSD  (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_DVSD        0x00000400          // Standard DV
Bluebook, DVPRO, DVSD
//! High Def DV codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_DV100       0x00000800
//! 8Bit Y'CrCb uncompressed video (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_UN8BIT              0x00001000
//! 10Bit Y'CrCb uncompressed video (gsGetSetValue::gsCompType)
```

```
#define GS_COMPTYPE_UN10BIT                 0x00002000
//! Panasonic HD to SDI codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_HDPAN           0x00010000
//! Sony HD to SDI codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_HDSONY                  0x00020000
//!Inverted 32 bit TGA
#define GS_COMPTYPE_BGRA_INVERT 0x00040000
//! Uncompressed RGB (DVS)
#define GS_COMPTYPE_ARGB            0x00100000
//! Uncompressed RGBA (DVS)
#define GS_COMPTYPE_RGBA            0x00200000
//! Uncompressed A BGR - TIFF
#define GS_COMPTYPE_ABGR            0x00400000
//! Uncompressed BGR A - BMP/TGA
#define GS_COMPTYPE_BGRA            0x00800000
//! Uncompressed RGB 24 Bit
#define GS_COMPTYPE_RGB                     0x00004000
//! Uncompressed BGR 24 Bit
#define GS_COMPTYPE_BGR                     0x00000020
//! Uncompressed Y'CrCb 4:2:2 (DVS, VG)
#define GS_COMPTYPE_YCRCB_422     0x01000000
//! Uncompressed Y'CrCb 4:2:2A (DVS, Dual VG)
#define GS_COMPTYPE_YCRCB_422A    0x02000000
//! Uncompressed Y'CrCb 4:4:4 (DVS, Dual VG)
#define GS_COMPTYPE_YCRCB_444     0x04000000
//! Uncompressed Y'CrCb 4:4:4A (DVS, Dual VG)
#define GS_COMPTYPE_YCRCB_444A    0x08000000
//! Uncompressed Y'CrCb 4:4:4A (DVS, Dual VG)
#define GS_COMPTYPE_YCRCB_422_STEREO 0x10000000
//! Uncompressed Y'CrCb 4:2:0
#define GS_COMPTYPE_YCRCB_420     0x20000000
//! DPX 10 bit rgb
#define GS_COMPTYPE_DPX_RGB10     0x40000000
//! DPX 10 bit YCbCr
#define GS_COMPTYPE_DPX_YCBCR10 0x00080000

#define GS_COMPTYPE_UNUSED_2      0x00008000
#define GS_COMPTYPE_UNUSED_7      0x80000000

/** @{ */
/** Standard Windows AVI container (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_AVI                      0x00000001
/** OpenDML AVI container (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
```

```
#define GS_SUPFILE_ODML                 0x00000002
/** Quicktime Mov/MooV container (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_QT                   0x00000004
/** Avid Open Media Format container (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_OMFI                 0x00000008
/** Drastic Fixed Frame container (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_FIX                  0x00000100
/** Audio only or separate audio formats (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_AUDONLY          0x00010000
/** Series of still file formats (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_STILLS               0x00100000
/** Other unspecified formats (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_UNK                  0x40000000
/** Any supported mediareactor format (gsGetSetValue::gsSupportedFileTypes
gsGetSetValue::gsIgnoreFileTypes)
* gsGetSetValue::gsRecFileFormat gsGetSetValue::gsRecAudFileFormat
gsGetSetValue::gsConvertFileFormat gsGetSetValue::gsConvertAudFileFormat */
#define GS_SUPFILE_ANY                  0x80000000
/** @} */

//! Allow playback or edit to edit output as necassary (gsGetSetValue::gsPBEE)
#define GS_PBEE_AUTO                    0x00000000
//! Allow playback only output - no passthough (gsGetSetValue::gsPBEE)
#define GS_PBEE_PB                      0x00000001
//! Device dependant output (gsGetSetValue::gsPBEE)
#define GS_PBEE_DEFAULT                 0x000000FF

//! Video servo reference auto (gsGetSetValue::gsServoRefSelect)
#define GS_SERVOREF_AUTO            0x00000000
//! Video servo reference external only (gsGetSetValue::gsServoRefSelect)
#define GS_SERVOREF_EXT                 0x00000001
```

```
//! Video servo reference device default (gsGetSetValue::gsServoRefSelect)
#define GS_SERVOREF_DEFAULT             0x000000FF

//! Use record/play head (gsGetSetValue::gsHeadSelect)
#define GS_HEADSEL_RECPLAY              0x00000000
//! Use play head (gsGetSetValue::gsHeadSelect)
#define GS_HEADSEL_PLAY                 0x00000001
//! (gsGetSetValue::gsHeadSelect)
#define GS_HEADSEL_DEFAULT              0x000000FF

//! Edit colour frame 2 field (gsGetSetValue::gsColorFrame)
#define GS_CLRFRM_2FLD                  0x00000000
//! Edit colour frame 4 field (gsGetSetValue::gsColorFrame)
#define GS_CLRFRM_4FLD                  0x00000001
//! Edit colour frame 8 field (gsGetSetValue::gsColorFrame)
#define GS_CLRFRM_8FLD                  0x00000002
//! Edit colour frame device default (gsGetSetValue::gsColorFrame)
#define GS_CLRFRM_DEFAULT               0x000000FF

//! Disable video reference (gsGetSetValue::gsVidRefDisable)
#define GS_VIDREF_DISABLE               0x00000000
//! Enable video reference (gsGetSetValue::gsVidRefDisable)
#define GS_VIDREF_ENABLE                0x00000001

//! Channel can play (video or audio or both) (gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_PLAY                 0x00000001
//! Channel can reverse play (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_REVPLAY              0x00000002
//! Channel can pause and display frame (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_PAUSE                0x00000004
//! Channel can jog below play speed (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_JOG                  0x00000008
//! Channel can shuttle above play speed (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_SHUTTLE              0x00000010
//! Channel can seek to any point (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_SEEK                 0x00000020
//! Channel can preview from in to out (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_PREVIEW              0x00000040
//! Channel has a stop mode (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_STOP                 0x00001000
//! Channel can pass through video (in stop) (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
```

```
#define GS_CHANCAP_ETOE                 0x00002000
//! Channel can record (video or audio or both) (gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_RECORD       0x00004000
//! Channel can edit from in to out (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_EDIT                 0x00008000
//! Channel can set clip name and prep record (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_RECSTOP              0x00010000
//! Channel can select recording channels (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_SELECTPRESET 0x00020000
//! Channel can eject the media (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_EJECT        0x00040000
//! Channel can play in a loop (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_LOOP                 0x00100000
//! Channel can display a vga preview (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_VGAPREVIEW   0x00200000
//! Channel can preview audio on a multi media card (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_AUDPREVIEW   0x00200000
//! Channel can play from a file (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_FILE                 0x01000000
//! Channel can play from a network feed (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_NET                  0x02000000
//! Channel can act like a clip space (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_CLIPSPACE    0x10000000
//! Channel can act like a vtr time code space (video or audio or both)
(gsGetSetValue::gsChanCapabilities)
#define GS_CHANCAP_TCSPACE              0x20000000
//! Channel can be configures as MPEG -- opens whole bunch of
//! settings on the options (specifically for Argus board right now).
#define GS_CHANCAP_MPEG_ENC             0x40000000
//! Do not use this bit - indicates error
#define GS_CHANCAP_ERROR        0x80000000
//! Channel can do anything except MPEG_ENC (by default this should not be)
#define GS_CHANCAP_ALL                  0x3FFFFFFF

//! Normal editing mode, no special speed compensation
cmdGetSetValue::gsSerialEditMode
#define GS_SERIALEDITMODE_NONE          0
//! Ignore all off speed play commands (CBS TimeLogic Mode)
cmdGetSetValue::gsSerialEditMode
```

```
#define GS_SERIALEDITMODE_IGNORE        1
//! Pause at each speed change, call time play when real play comes (CTV mode)
cmdGetSetValue::gsSerialEditMode
#define GS_SERIALEDITMODE_FAKE          2

//! cmdGetSetValue::gsProxyMode Do not automatically proxy anything
#define GS_PROXYMODE_NOTHING            0x000000
//! cmdGetSetValue::gsProxyMode Proxy and file that is opened (for read/write/check)
#define GS_PROXYMODE_EVERYTHING         0x000002
//! cmdGetSetValue::gsProxyMode Proxy files while they are recording (with supported
source types)
#define GS_PROXYMODE_RECORD                     0x000001
//! cmdGetSetValue::gsProxyMode Proxy files once they have finished recording
#define GS_PROXYMODE_AFTERRECORD        0x000010
//! cmdGetSetValue::gsProxyMode       Abort all active proxies
#define GS_PROXYMODE_ABORTALL           0x0ffff0

// WaveForm, Vectorscope control

//! cmdGetSetValue::gsWaveVectorSetup standard picture
#define GS_WAVEVECTOR_PICTURE                   0x00000001
//! cmdGetSetValue::gsWaveVectorSetup standard vectorscope
#define GS_WAVEVECTOR_VECTORSCOPE               0x00000002
//! cmdGetSetValue::gsWaveVectorSetup standard waveform
#define GS_WAVEVECTOR_WAVEFORM                  0x00000004
//! cmdGetSetValue::gsWaveVectorSetup parade RGB waveform
#define GS_WAVEVECTOR_WAVEFORM_RGB              0x00000008
//! cmdGetSetValue::gsWaveVectorSetup parade Y CB CR waveform
#define GS_WAVEVECTOR_WAVEFORM_YCBCR0x00000010
//! cmdGetSetValue::gsWaveVectorSetup histogram
#define GS_WAVEVECTOR_HISTOGRAM                         0x00000020
//! cmdGetSetValue::gsWaveVectorSetup parade histogram
#define GS_WAVEVECTOR_HISTOGRAM_SEP             0x00000040
//! cmdGetSetValue::gsWaveVectorSetup illegal colors
#define GS_WAVEVECTOR_ILLEGAL                   0x00000100
//! cmdGetSetValue::gsWaveVectorSetup illegal colors
#define GS_WAVEVECTOR_DATA                              0x00001000

//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel RED
#define GS_WAVEVECTOR_CHANNEL_R                 0x00000001
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel GREEN
#define GS_WAVEVECTOR_CHANNEL_G                 0x00000002
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel BLUE
#define GS_WAVEVECTOR_CHANNEL_B                 0x00000004
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel ALPHA
#define GS_WAVEVECTOR_CHANNEL_A                 0x00000008
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel Y (Luma)
#define GS_WAVEVECTOR_CHANNEL_Y                 0x00000010
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel CB
```

```
#define GS_WAVEVECTOR_CHANNEL_CB         0x00000020
//! cmdGetSetValue::gsWaveVectorSetup dwStart color channel CR
#define GS_WAVEVECTOR_CHANNEL_CR         0x00000040


//! Command is not supported see cmdType::ctGetValue, cmdType::ctSetValue,
cmdType::ctValueSupported
#define GS_NOT_SUPPORTED 0xFFFFFFFFUL
/**
*  Parameter is bad see cmdType::ctGetValue, cmdType::ctSetValue,
cmdType::ctValueSupported and
* MEDIACMD::dwPosition, MEDIACMD::dwStart and MEDIACMD::dwEnd
*/
#define GS_BAD_PARAM          0xFFFFFFFEUL
/**
* False for boolean cmdType::ctGetValue, cmdType::ctSetValue
*/
#define GS_FALSE       0x00
/**
* True for boolean cmdType::ctGetValue, cmdType::ctSetValue
*/
#define GS_TRUE                0x01
/**
* Disable a feature or command
*/
#define GS_DISABLE    0x00
/**
* Enable a feature or command
*/
#define GS_ENABLE     0x01
/**
* Default for cmdType::ctGetValue, cmdType::ctSetValue (usually in relation to VTR
setup)
*/
#define GS_DEFAULT   0xFF     /* Default set by user on device */
//! Use field cmdType::ctGetValue, cmdType::ctSetValue (for pause/freeze as opposed to
frame)
#define GS_FIELD       0x00     /* Frame or Field one (gs dependant) */
//! Use field 1 cmdType::ctGetValue, cmdType::ctSetValue (for record/playback starts
and edits)
#define GS_FIELD1      0x01     /* Field 1 - norm editing / single (1) freeze */
//! Use field 2 cmdType::ctGetValue, cmdType::ctSetValue (for record/playback starts
and edits)
#define GS_FIELD2      0x02     /* Field 2 - off editing / single (2) freeze */
//! Use frame cmdType::ctGetValue, cmdType::ctSetValue (for pause/freeze as opposed
to field)
#define GS_FRAME       0x03     /* Freeze frame */
//! Set value to unity (levels, tbc) or default (compression type, amount)
#define GS_UNITY        0xFFFFFFFF
```

```
//! AvHAL input set normal SDI or Analog single link
#define GS_ALPHACHROMA_SINGLE     0x01
//! AvHAL input set normal SDI plus a Y only SDI alpha plane
#define GS_ALPHACHROMA_ALPHA      0x02
//! Dual Link or HSDL input setup (2 HD-SDI 4:4:4 combined)
#define GS_ALPHACHROMA_DUAL               0x04

//! Supports 8 bits per pixel component (normally YCbCr, for RGB see below)
#define GS_BITCOUNT_8                     0x01
//! Supports 10 bits per pixel component (normally YCbCr, for RGB see below)
#define GS_BITCOUNT_10                    0x02
//! Supports 3 (RGB) 8 bit components per pixel
#define GS_BITCOUNT_24                    0x04
//! Supports 3 (RGB) 10 bit components per pixel (e.g. standard DPX)
#define GS_BITCOUNT_30                    0x08
//! Supports 4 (RGBA) 8 bit components per pixel (e.g. standard TGA)
#define GS_BITCOUNT_32                    0x10
//! Supports YCbCr 4:2:0 (YUV) 8 bit components (e.g. i420, yv12)
#define GS_BITCOUNT_12                    0x20    /* 4:2:0 */


// This is used to send a command to the system from any driver
// Also used for returning current state
// e.g. RS-422, RS-232, Keyboard, User Interface, etc.
#pragma pack(4)

typedef struct /*tagCMD_QUEUE_ELEM*/ {
        // NOTE:  Must match D_LNODE
#ifndef __midl
        //! INTERNAL dlist.dll Link List Pointers - do not use
  void * pPrev; // Next lnode
        //! INTERNAL dlist.dll Link List Pointers - do not use
  void * pNext; // Prev lnode
        //! INTERNAL dlist.dll Link List Pointers - do not use
  void * pList;  // Parent or List owner
#else
  DWORD pPrev, pNext, pList;
#endif
        /**
        * The command identifier is used to confirm that the command is properly
formatted and
        * of a version that the receiver understands.  This member should always be set
to #MEDIACMD_CURRENT
        */
        DWORD                   dwCmdID;
        /**
        * This member contains the entire size of the structure being sent.  Certain
commands may not
```

```
        * require all fields to be completely understood.  Most commands will send the
bulk of the
        * structure and remove only unused parts of arbID[], though this should not be
counted on.
        * It may be assumed that all commands will include the #pPrev, #pNext, #pList,
#dwCmdID,
        * #dwStructSize, #dwChannel, #ctCmd, and #cfFlags members with every
command so dwStructSize
        * must be at least 32 bytes in length.
        */
        DWORD                   dwStructSize;
        /**
        * The target channel.  For direct connect channels (eg. in the same machine)
this member
        * is ignored.  For serial/network/piped channels, this allows a single transport to
        * access multiple channels.  It is also used for kernel<->user mode commands
for
        * DComCtrl.sys and vvwCtl.dll as well as vvwNet.dll.
        */
        DWORD                   dwChannel;              //
        /**
        * The ctCmd member contains the basic or overall command.  It is of the type
#cmdType.
        * It includes transport and setup commands that may be immediate or delayed
depending
        * on the rest of the structure.  Basic commands include:  cmdType::ctPlay,
        * cmdType::ctStop, cmdType::ctPause (or seek), cmdType::ctRecord,
cmdType::ctGetState,
        * cmdType::ctGetValue and cmdType::ctSetValue.
        */
        enum cmdType ctCmd;                     // The command - enum added for ansi
'C' compiles
        /**
        * The cfFlags member contains flags that modify the operation of the other
structure
        * members.  These flags are of the type #cmdFlags.  The flags normally specify
which
        * other members are to be used for the command as well as modifiers for
delaying
        * or otherwise augmented commands.  Basic flags include:
cmdFlags::cfDeferred,
        * cmdFlags::cfUseSpeed, cmdFlags::cfUsePosition, cmdFlags::cfUseStart,
cmdFlags::cfUseEnd,
        * cmdFlags::cfUsePositionOffset, cmdFlags::cfUseClipID.
        */
        DWORD                   cfFlags;        // Command flags
        /**
        * This member controls the speed of a command.  Normally it is used with
cmdType::ctPlay and
```

```
        * required cmdFlags::cfUseSpeed to be set in the #cfFlags member to be used.
The defines
        * #SPD_FWD_PLAY, #SPD_PAUSE, #SPD_REV_PLAY, #SPD_FWD_MAX,
#SPD_REV_MAX can be used, or any
        * other valid speed.  This table outlines the basic linear speed changes.
        \code
   SPD_REV_MAX     SPD_REV_PLAY    SPD_PAUSE       SPD_FWD_PLAY
SPD_FWD_MAX
   -5896800        -65520          0               65520           5896800
   -90x            -1x             0               1x              90x
   -9000%          -100%           0               100%            90000%
   -90.0           -1.0            0               1.0             90.0

   Some Normal Speeds (for reverse, set to minus)
   10 times play   10.0    1000%   655200
   5 times play    5.0     500%    327600
   2 times Play    2.0     200%    131040
   Play Normal     1.0     100%    65520
   Two Third       0.66    66%     43680
   Half Play       0.5     50%     32760
   One Third       0.33    33%     21840

   note:  Speed table is linear (log like serial control must be converted)
        \endcode
        */
        LONG            lSpeed;                 // '-'Reverse '0'pause '+'forward
        /**
        * VIDEO Bit array of channels where<BR>
        * 1 = first channel<BR>
        * 2 = second channel<BR>
        * 4 = third channel<BR>
        * 8 = fourth channel<BR>
        * Also, #cmdVidChan enum may be used
        */
        DWORD                   dwVideoChannels;// Video channels the cmd involves
        /**
        * AUDIO Bit array of channels where<BR>
        * 1 = first channel<BR>
        * 2 = second channel<BR>
        * 4 = third channel<BR>
        * 8 = fourth channel<BR>
        * Also, #cmdAudChan enum may be used
        */
        DWORD                   dwAudioChannels;// Audio channels the cmd involves
        /**
        * INFO Bit array if channels.  See #cmdinf for possible channels
        * including cmdinf::Ltc, cmdinf::Vitc, cmdinf::Copyright, etc.
        */
        DWORD                   dwInfoChannels; // Info channels the cmd involves
```

```
        /**
        * This member may have many meaning depending on the rest of the
        * MEDIACMD structure.  <BR>
        * if #ctCmd is cmdType::ctGetValue, cmdType::ctSetValue or
cmdType::ctValueSupported then it contains the #cmdGetSetValue to use
        * if #ctCmd is cmdType::ctTransfer it contains the source channel for the
transfer (the command is always set to the target)
        * if #cfFlags includes cmdFlags::cfUseCmdAlt and cmdFlags::cfTimeMs is
contains a millisecond version of the performance clock
        * if #ctCmd is cmdType::ctRecord then it may be used as a millisecond offset to
start of record
        * if #ctCmd is cmdType::ctPlay then it may be used as a millisecond offset to
start of playback
        */
        DWORD           dwCmdAlt;       // Time delay, alternate channel
        /**
        * For most commands, this will be the current or target frame counter position
for the command.<BR>
        * Check of cmdFlags::cfUsePosition or cmdFlags::cfUsePositionOffset (becomes
long against current position) to make sure this member is valid.<BR>
        * For cmdType::ctGetValue, cmdType::ctSetValue or
cmdType::ctValueSupported this is primary set and return member.
        */
        DWORD           dwPosition;     // Current or third edit point
position
        /**
        * For most commands, this will be the starting frame counter position for the
command.<BR>
        * Check of cmdFlags::cfUseStart or cmdFlags::cfUseStartOffset (becomes long
against current position) to make sure this member is valid.<BR>
        * For cmdType::ctGetValue, cmdType::ctSetValue or
cmdType::ctValueSupported this is secondary set and return member.
        */
        DWORD           dwStart;        // Command start point in
frames,
        /**
        * For most commands, this will be the ending frame counter position for the
command.<BR>
        * Check of cmdFlags::cfUseEnd or cmdFlags::cfUseEndOffset (becomes long
against current position) to make sure this member is valid.<BR>
        * For cmdType::ctGetValue, cmdType::ctSetValue or
cmdType::ctValueSupported this is secondary set and return member.
        */
        DWORD           dwEnd;          // Command end point
(exclusive) in frames
        /**
        * Free form memory area mostly used for string and clip handling. Valid if the
        * #cfFlags member includes cmdFlags::cfUseClipID
```

```
	* Basic Clip Name:	first 8 BYTES alpha number, may include spaces,
terminated by 0<BR>
	* Two Clip Names:  'Basic Clip Name' followed immeadiately by another 'Basic
Clip Name'
	* Clip+File Name:  'Basic Clip Name' followed by a null terminated file/path
name.
	* String:		Null terminated ANSI string.
	*/
	BYTE		arbID[CMD_MAX_CLIP_ID_LEN+2+2];  // Odetics/Louth
Identifier
} MEDIACMD, *PMEDIACMD;

//! Old name, use MEDIACMD instead
#define CmdQueueElem MEDIACMD
//! Old name, use PMEDIACMD instead
#define pCmdQueueElem PMEDIACMD

//! Declare a media cmd structure to all pause
#define DECLARE_MEDIACMD(__mCmd_) MEDIACMD __mCmd_ =	\
	{NULL, NULL, NULL,
	\
	MEDIACMD_CURRENT, (DWORD)sizeof(MEDIACMD), CHAN_ILLEGAL,	\
	ctPause, 0UL, SPD_ILLEGAL,					\
	(DWORD)vidChanAll, (DWORD)audChanAll,				\
	(DWORD)infChanAll, (DWORD)gsNextClip,				\
	(DWORD)TC_ILLEGAL, (DWORD)TC_ILLEGAL, (DWORD)TC_ILLEGAL,	\
	{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\

			\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		\
	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	\
	}};
//! Initailize a media cmd structure to all illegal (no command)
#ifdef _WIN32

#define INIT_MEDIACMD(__mCmd_)	{					\
```

```
        ZeroMemory(&__mCmd_, sizeof(MEDIACMD));                         \
        __mCmd_.dwCmdID = MEDIACMD_CURRENT;                                     \
        __mCmd_.dwStructSize = sizeof(MEDIACMD);                \
        __mCmd_.dwChannel = CHAN_ILLEGAL;                                       \
        __mCmd_.lSpeed = SPD_ILLEGAL;                                           \
        __mCmd_.dwVideoChannels = (DWORD)vidChanAll;            \
        __mCmd_.dwAudioChannels = (DWORD)audChanAll;            \
        __mCmd_.dwInfoChannels = (DWORD)infChanAll;                     \
        __mCmd_.dwCmdAlt = 0;
        \
        __mCmd_.dwPosition = TC_ILLEGAL;                                \
        __mCmd_.dwStart = TC_ILLEGAL;                                           \
        __mCmd_.dwEnd = TC_ILLEGAL;       }
#else
#define INIT_MEDIACMD(__mCmd_)   {                                              \
        memset(&__mCmd_,0, sizeof(MEDIACMD));                           \
        __mCmd_.dwCmdID = MEDIACMD_CURRENT;                                     \
        __mCmd_.dwStructSize = sizeof(MEDIACMD);                \
        __mCmd_.dwChannel = CHAN_ILLEGAL;                                       \
        __mCmd_.lSpeed = SPD_ILLEGAL;                                           \
        __mCmd_.dwVideoChannels = (DWORD)vidChanAll;            \
        __mCmd_.dwAudioChannels = (DWORD)audChanAll;            \
        __mCmd_.dwInfoChannels = (DWORD)infChanAll;                     \
        __mCmd_.dwCmdAlt = 0;
        \
        __mCmd_.dwPosition = TC_ILLEGAL;                                \
        __mCmd_.dwStart = TC_ILLEGAL;                                           \
        __mCmd_.dwEnd = TC_ILLEGAL;       }
#endif
//! Initailize a media cmd pointer to all illegal (no command)
#ifdef _WIN32
#define INIT_PMEDIACMD(__mCmd_)  {
        \
        ZeroMemory(__mCmd_, sizeof(MEDIACMD));                                  \
        ((PMEDIACMD)__mCmd_)->dwCmdID = MEDIACMD_CURRENT;               \
        ((PMEDIACMD)__mCmd_)->dwStructSize = sizeof(MEDIACMD);\
        ((PMEDIACMD)__mCmd_)->dwChannel = CHAN_ILLEGAL;                 \
        ((PMEDIACMD)__mCmd_)->lSpeed = SPD_ILLEGAL;                             \
        ((PMEDIACMD)__mCmd_)->dwVideoChannels = (DWORD)vidChanAll;      \
        ((PMEDIACMD)__mCmd_)->dwAudioChannels = (DWORD)audChanAll;      \
        ((PMEDIACMD)__mCmd_)->dwInfoChannels = (DWORD)infChanAll;       \
        ((PMEDIACMD)__mCmd_)->dwCmdAlt = CHAN_ILLEGAL;                  \
        ((PMEDIACMD)__mCmd_)->dwPosition = TC_ILLEGAL;                  \
        ((PMEDIACMD)__mCmd_)->dwStart = TC_ILLEGAL;                             \
        ((PMEDIACMD)__mCmd_)->dwEnd = TC_ILLEGAL;      }
#else
#define INIT_PMEDIACMD(__mCmd_)  {
        \
        memset(__mCmd_, 0,sizeof(MEDIACMD));                                   \
```

```
        ((PMEDIACMD)__mCmd_)->dwCmdID = MEDIACMD_CURRENT;                \
        ((PMEDIACMD)__mCmd_)->dwStructSize = sizeof(MEDIACMD);\
        ((PMEDIACMD)__mCmd_)->dwChannel = CHAN_ILLEGAL;                  \
        ((PMEDIACMD)__mCmd_)->lSpeed = SPD_ILLEGAL;                      \
        ((PMEDIACMD)__mCmd_)->dwVideoChannels = (DWORD)vidChanAll;       \
        ((PMEDIACMD)__mCmd_)->dwAudioChannels = (DWORD)audChanAll;       \
        ((PMEDIACMD)__mCmd_)->dwInfoChannels = (DWORD)infChanAll;        \
        ((PMEDIACMD)__mCmd_)->dwCmdAlt = CHAN_ILLEGAL;                   \
        ((PMEDIACMD)__mCmd_)->dwPosition = TC_ILLEGAL;                   \
        ((PMEDIACMD)__mCmd_)->dwStart = TC_ILLEGAL;                      \
        ((PMEDIACMD)__mCmd_)->dwEnd = TC_ILLEGAL;      }
#endif


#pragma pack()

//! SizeOf a command queue without the arbID
#define CMD_QUEUE_ELEMSIZE ((size_t)(&((pCmdQueueElem)(0))->arbID[0]))
//! SizeOf basic mediacmd structure without any clip id
#define SIZEOF_MEDIACMD_BASE CMD_QUEUE_ELEMSIZE
//! SizeOf mediacmd structure with a 8 byte clip id and terminating 0
#define SIZEOF_MEDIACMD_CLIPID (CMD_QUEUE_ELEMSIZE + 9)
//! SizeOf a complete mediacmd structure
#define SIZEOF_MEDIACMD sizeof(MEDIACMD)

/**
* \page mediacmdfastinfo MEDIACMD Fast Info Page

  \section mediamcmdinto MEDIACMD Introduction

<BR>This document covers inter-device communication protocol used for control and information exchange across a Drastic Media machine, Intranet (LAN) or Internet (WAN). This document does not include information concerning low-level file structures, raw data access, or hardware specific variations.
<BR>
<BR>Drastic uses an inter-process piping metaphor to allow the implementation of media distribution and control over a small or large sized installation.  A media distribution system normally includes the following components:
<BR>
\li Receiving and interpreting external time code based instructions (Sony/SMPTE/VDR serial protocols)
\li Receiving and implementing external file or clip based instructions (Odetics/Alamar)
\li Configuration and local control (Graphical User Interface)
\li Storage management (Clip/File/Time Code/LTC/VITC/User)
\li Transport control (On time/Pre load/Pathing/QOS)
\li Hardware audio/video/compression control (Local/Effective Remote)
<BR>
<BR>Each of these components must be able to intercommunicate with one or more other components to maintain control and status information within the processes or
```

systems. To allow transportation of these command elements across a local or wide area network, a protocol and transport method compatible with the running platforms and the networks connecting them must be used.

Once the protocol is in place, each of the drivers within a machine or network must have an established command set to inter communicate. The object of each component of the system is to receive media commands, send media commands or control media based on those commands. This requirement is met by the drastic MediaCmd structure, which is used for communication between all devices.

Physical Transport

The physical mechanisms for transporting commands throughout a system are encapsulated in the Drastic VVWNet library. The connection is in some ways similar to a Unix or NT, but does not actually use pipes. The VVWNet provides a simple interface for sending command packets between drivers within the same machine or different machines attached by a network. The VVWNet may use any underlying network protocol, but currently supports TCP/IP and IPX.

\section mediacmdlinks MEDIACMD QuickLinks
<CODE>

 The structure: MEDIACMD

 The commands: #cmdType

 The Flags: #cmdFlags

 Channels: #cmdVidChan, #cmdAudChan, #cmdinf

 GetSet Commands #cmdGetSetValue

  Basic cmdGetSetValue::gsTc

  Clip cmdGetSetValue::gsGetNextClip

  Channel cmdGetSetValue::gsAudChan

  Audio cmdGetSetValue::gsAudInSelect

  Video General cmdGetSetValue::gsVidFreeze

  Video Input cmdGetSetValue::gsVidInSelect

  Video TBC cmdGetSetValue::gsVidSetup

  Video Output cmdGetSetValue::gsVidOutSelect

  Signal Type cmdGetSetValue::gsSignalFormat

  Compression cmdGetSetValue::gsCompType

  Storage cmdGetSetValue::gsTotalStorageAvail

  Control cmdGetSetValue::gsLocal

  Info cmdGetSetValue::gsVVWVersion

  Internal cmdGetSetValue::gsSetHwnds

  Mode+Info cmdGetSetValue::gsChannelExist

  Returns #GS_NOT_SUPPORTED

 Declaration Local #DECLARE_MEDIACMD

 Init Local #INIT_MEDIACMD

 Init Pointer #INIT_PMEDIACMD

 Sizes #SIZEOF_MEDIACMD_BASE

</CODE>
\section mediacmdsamplecmds MEDIACMD Sample Commands

The following are sample commands as sent through media command. The samples are taken from the VVW series of products, and as such are guaranteed to work with them. Other OEMs and manufacturers supporting the MediaCmd interface may vary from this specification.

\b Simple \b Play
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctPlay

\b Simple \b Pause
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctPause

\b Simple \b Stop
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctStop

\b Play \b From-To
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctPlay
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd
MEDIACMD::dwStart = 1800 // 1 minute in frames NTSC NDF
MEDIACMD::dwEnd = 2100 // 1 minute 10 seconds NTSC NDF

\b Play \b DMC \b (Play \b at \b speed)
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 36
MEDIACMD::ctCmd = #ctPlay
MEDIACMD::cfFlags = #cfUseSpeed
MEDIACMD::lSpeed = 32760 // Half play speed forward

\b Record \b Edit
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctRecord
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfUsePresets
MEDIACMD::dwVideoChannels = 0x01 // Record V
MEDIACMD::dwAudioChannels = 0x03 // Record A1 & A2 (AA)
MEDIACMD::dwInfoChannels = 0x00
MEDIACMD::dwStart = 3000
MEDIACMD::dwEnd = 3200

\b Play \b Two \b Segments
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT

MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctPlay
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd
MEDIACMD::dwStart = 2200
MEDIACMD::dwEnd = 2500

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctPlay
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfDeferred
MEDIACMD::dwStart = 5300
MEDIACMD::dwEnd = 5450

\b Play \b A \b Named \b Clip
Char szName = "C:\Adir\Afile.OMF"
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60 + strlen(szName)
MEDIACMD::ctCmd = #ctPlay
MEDIACMD::cfFlags = #cfUseClipID
strcpy(MEDIACMD::arbID , szName)

\b Seek \b To \b 1 \b Minute
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 48
MEDIACMD::ctCmd = #ctPause
MEDIACMD::cfFlags = #cfUsePosition
MEDIACMD::dwPosition = 1800 // Frames NTSC NDF

\b Step \b Reverse \b 1 \b Frame
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 48
MEDIACMD::ctCmd = #ctPause
MEDIACMD::cfFlags = #cfUsePositionOffset
MEDIACMD::dwPosition = (DWORD) -1

\b Record \b A \b 1 \b Minute \b Named \b Clip
Char szName = "C:\Adir\ArecordFile.MOV"
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60 + strlen(szName)
MEDIACMD::ctCmd = #ctRecord
MEDIACMD::cfFlags = #cfUseClipID | #cfUseEnd
MEDIACMD::dwEnd = 1800
strcpy(MEDIACMD::arbID , szName)

\b Record \b An \b Odetics \b or \b Louth \b Clip \b (ext)
(Note: The stop command is required for accuracy on some DDRs)
Char szName = "THISCLIP" // Max 8 char
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60 + strlen(szName)

MEDIACMD::ctCmd = #ctStop // Record ready
MEDIACMD::cfFlags = #cfUseClipID
strcpy(MEDIACMD::arbID , szName)

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60 + strlen(szName)
MEDIACMD::ctCmd = #ctRecord
MEDIACMD::cfFlags = #cfUseClipID
strcpy(MEDIACMD::arbID , szName)

\b Eject \b The \b Current \b Media
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctEject

\b Setup \b The \b Video \b To \b Nominal
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctSetValue
MEDIACMD::dwCmdAlt = gsVidSetup
MEDIACMD::dwStart = 0

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctSetValue
MEDIACMD::dwCmdAlt = gsVidVideo
MEDIACMD::dwStart = 0

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctSetValue
MEDIACMD::dwCmdAlt = gsVidHue
MEDIACMD::dwStart = 0

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctSetValue
MEDIACMD::dwCmdAlt = gsVidChroma
MEDIACMD::dwStart = 0

\b Check \b The \b Device \b Type
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctGetValue
MEDIACMD::dwCmdAlt = gsVtrType
MEDIACMD::dwStart = 0
(Returns: Vtr/Ddr/Media type in .dwStart)

\b Change \b The \b Default \b TC \b Type \b To \b VITC

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctSetValue
MEDIACMD::dwCmdAlt = gsVitcTc
MEDIACMD::dwStart = GS_DEFAULT

\b Transfer \b From \b External \b VTR \b To \b Internal \b Channel
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctTransfer
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd | #cfUsePresets
MEDIACMD::dwVideoChannels = 0x01 // Cature Video
MEDIACMD::dwAudioChannels = 0x00 // No Audio
MEDIACMD::dwInfoChannels = 0x00
MEDIACMD::dwCmdAlt = hExternalChannel // The VTR channel
MEDIACMD::dwPosition = 2100 // Where to record to
MEDIACMD::dwStart = 10850 // Source In on VTR
MEDIACMD::dwEnd = 11000 // Source Out on VTR

\b Insert \b Media \b From \b File \b Over \b Current
Char szName = "C:\Adir\ArecordFile.MOV"
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60 + strlen(szName)
MEDIACMD::ctCmd = #ctInsert
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd | #cfUseClipID
MEDIACMD::dwPosition = 2100 // Insert @ on target channel
MEDIACMD::dwStart = 0 // Start on source file
MEDIACMD::dwEnd = 180 // End on source file
strcpy(MEDIACMD::arbID , szName) // Name of source file

\b Delete \b A \b Section \b Of \b Media \b (No \b Hole)
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctDelete
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfRipple // Remove #cfRipple to leave hole
MEDIACMD::dwStart = 140500 // Start on target media
MEDIACMD::dwEnd = 150010 // End on target media

\b Trim \b A \b Clip
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 60
MEDIACMD::ctCmd = #ctTrim
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd
MEDIACMD::dwPosition = 2100 // Clip @ position
MEDIACMD::dwStart = +32 // Clip 32 frames from start
MEDIACMD::dwEnd = (DWORD) -12 // Clip 12 frame from end

\b Terminate \b Session \b (Restart \b At \b Default)
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctTerminate

\b Abort \b Current \b Command
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT
MEDIACMD::dwStructSize = 28
MEDIACMD::ctCmd = #ctAbort

* \section mediacmdsamplertns MEDIACMD Sample Returns
Sample Returns

MEDIACMD::dwCmdID == #MEDIACMD_CURRENT
MEDIACMD::dwStructSize == #SIZEOF_MEDIACMD
MEDIACMD::ctCmd == cmdType::ctPlay
MEDIACMD::cfFlags == 0
\b Normal \b Play \b (100% \b Play \b Speed)

MEDIACMD::dwCmdID == #MEDIACMD_CURRENT
MEDIACMD::dwStructSize == #SIZEOF_MEDIACMD
MEDIACMD::ctCmd == cmdType::ctRecord
MEDIACMD::cfFlags == 0
\b Normal \b Record \b (Crash \b Record)

*/

/*! \mainpage MediaCmd SDK
*
* \section intro_sec Introduction
*
The Drastic MediaCmd SDK is the mechanism by which all the elements of Drastic's DDRs communicate with one and other. This includes:

* Controlling VVW DDR Servers, QuickClip locally, and QuickClip with network option remotely
* Controlling 9 pin serial VTRs and Servers via Sony, Odetics or VDCP protocol
* Receiving commands from 9 pin serial controller via Sony, Odetics or VDCP protocol
* Receiving commands from Drastic GUIs, servers and controllers
* Building HTML/Ajax status and control pages

MediaCmd is the commincation method used within Drastic's DDR products. Any operation you see in a Drastic interface is available to your application through MediaCmd.

* \section overview_sec Overview

MediaCmd is a simple structure that supports a small, well defined set of commands for communicating transport, status and setup information between components in Drastic's DDR software. There are a number of fields in the stucture, but the important fields are:

* ctCmd – the primary command of this packet (Play, Pause, Stop, Record, etc)
* lSpeed – the transport speed for any play commands (integer where 65520 = normal forward play)
* dwPosition – the frame position for any play, pause or record commands
* dwStart – the starting frame for any play or record commands (inclusive)
* dwEnd – the ending frame for any play or record commands (exclusive)
* arbID – clip name, file name or other string/binary data for the command
* cfFlags – denotes which fields above are valid and their meaning

With the standard initialization of the structure, you can quickly build commands in this structure by changing a few members and sending it. The primary motion commands are ctPlay, ctPause, ctStop, ctRecStop, ctRecord, ctEject and ctTransfer. To get the current state (position, speed, start and end, current clip), the command ctGetState will return a filled in MediaCmd. For setup and less common status (e.g. video input, audio rms level, genlock) there is ctGetValue and ctSetValue. This is documented in the Low Level Header Docs.

Hopefully, you will not have to deal with the MediaCmd structure directly. The SDK includes a series of simple commands that should provide 99% of what your application needs. These functions are simply wrappers that create and send MediaCmd structures. The source for all these functions is provided in the SDK under SRC/General/vvwIF.cpp in case you need to modify or create new commands. The commands have slightly different names depending on which interface you use, but have the same root name, such as: Play(), PlayFromTo(), Stop(), Pause(), Seek(), Record() and UpdateStatus(). Commands are also included for getting clip lists (GetNextClip()) and EDL elements from ::VTR_TC time code spaces (EDLResetToStart(), EDLGetEdit()). A selection of the most common settings are also included (SetVideoInput(), SetAudioInput(), SetVideoGenlock(), GetAudioPeakRMS(), etc). This interface is documented in the MediaCmd Documentation (previously called "VVW Interface Specification").

* \section installation_sec Installation

To properly work with the MediaCmd SDK, you should have a copy of the QuickClip software installed on your development system. Even if you target application will only use a part of the QuickClip software, it should all be installed for the development phase. Before proceeding with the SDK you should familarize your self with QuickClip's operation and toolset. All the elements available within QuickClip are the same elements available to your application through the SDK.
Once you have QuickClip installed, you should install the MediaCmd SDK. This will install the headers, libraries and source needed to control QuickClip from your application.

* \section accessmethod_sec Choosing An Access Method

The SDK access method you should use depends on what you you would like your application to do, what programming language you are using and how involved you would like to/need to get in the low level MediaCmd structures. No matter which method you choose, the MediaCmd structure packets are exactly the same. Here are the main access method, with their pros and cons:

ActiveX
Type: Microsoft ActiveX/COM access method
Pros: Easy to program, 1:1 relationship with QuickClip/XO interface.
Cons: Uses same config as QuickClip/XO. Requires a local copy of QuickClip.
Setup: Register VVW.DLL using RegSvr32.exe in the QuickClip installation directory.
Issues: Difficult to use when communicating via TCP/IP within the same machine. Can be overcome by using the default pipe communication system, but this requires changes for remote network control.

Direct Link
Type: Direct link to VVW.DLL
Pros: No ActiveX layer, code compatible with Linux, Irix, Mac OS-X.
Cons: Uses default config from QuickClip/XO, application must be run in QuickClip directory. Requires a local copy of QuickClip.
Setup: Link to vvw.lib, include vvw.h. Copy application into the QuickClip directory before running
Issues: Needs access to VVW.dll and all its support DLLs/D1Xs. Still needs to be setup by LocalConfig.exe or QuickClip/XO

Network DLL
Type: Direct line to vvwNet2.dll
Pros: Consistant interface between local/remote and various OSs. Does not require a local copy of QuickClip.
Cons: Requires vvwNet2.dll and support dlls
Setup: Link to vvwNet2.lib, include vvwNet2.h. Copy dll set from SDK/bin directory with your application
Issues: Use the netOpenLocal function to avoid QuickClip configuration issues. Requires a few dlls to be added to you application installations. Does not run the client software automatically, so you application may need to start it, depending on what your application is doing.

Network Direct
Type: Direct compile of network sources in your app or your dll.
Pros: No extra dlls. Easy to customize and modify. Lots of comands already written.
Cons: You app needs to handle setup and may need to run QuickClip.exe/VVWServer.exe/QCRun.exe.
Setup: Copy source files from vvwNet2 into you project, modify and compile
Issues: Does not run the client software automatically, so you application may need to start it, depending on what your application is doing.

Manual
Type: Use the structures and defines to write your own communication and control layer.

Pros: This is required if you are using an unsupported development platform like PHP.
Cons: Everything has to be built and tested from the ground up.
Setup: None.
Issues: Unless you absolutely have to, this method is not recommended.

* \section sdkstructure_sec SDK Structure

The location of the SDK directories will depend on the location you choose during the installation, but the directories within there will always be the same:

* /BIN – Copies of the minimum dll set from a QuickClip installation.
* /LIB – Libraries required to link the vvwNet2.dll, examples and your application
* /INC – Header files required to compile vvwNet2.dll, examples and your application
* /Src/vwwNet2 – The source to our vvwNet2.dll from QuickClip
* /Src/General – Useful source files that do not compile into examples directly. The most important would be vvwIF.cpp that is the code behind the SDK functions described below.
* /Sample – Broken down into sub directories based on access type
    - /ActiveX – Examples that use the ActiveX control
    - /Direct – Examples that link directly to DLLs
    - /Java – Java based examples
    - /HTTP – Ajax based examples (must use QuickClip HTTP server to run)

* \section maindoclink_sec Main Documentation Links

*** PDF version of the MediaCmd Documentation
http://www.drastictech.com/manuals/VVW%20Interface%20Specification.pdf

*** Online version of the MediaCmd Documentation
http://www.drastictech.com/manuals/VVW%20Interface%20Specification.html

* \section lowlevellink_sec Low Level Header Documentation Links

*** Windows CHM help file version of the MediaCmd headers
http://www.drastictech.com/manuals/MediaCmd.chm

*** Online version of the MediaCmd headers
http://www.drastictech.com/manuals/mediacmd/

* \section htmlajaxlink_sec HTTP XML AJAX Documentation Links

*** Wikki area for HTTP XML MediaCmd
http://www.drasticpreview.org/wakka.php?wakka=DrasticHttpCommands&v=d9c
*

```
 *
 */
#endif //_MEDIACMD_INCLUDED_H
```

Appendix II – MediaCmdX.h

Please see help file MediaCmd.chm (JavaDoc)

Appendix III – VVWIF.h

```
/*****************************************************************************
*
 * $Id$:
 *
 * $HeadURL$:
 * $Author$:
 * $Revision$:
 * $Date$:
 *
 * Copyright © 1998-2007 Drastic Technologies Ltd.  All Rights Reserved.
 * 523 The Queensway, Suite 102 Toronto ON M8V 1Y7
 * 416 255 5636  fax 255 8780
 * engineering@drastictech.com      http://www.drastictech.com
 *****************************************************************************/

#ifdef _WIN32
        #include <direct.h>
        #ifndef _CRT_SECURE_NO_WARNINGS
                #define _CRT_SECURE_NO_WARNINGS
        #endif
#else
        #define BOOL bool
        #define DWORD unsigned long
        #include <string.h>
        #include <sys/timeb.h>
        #include "VVWTCPIP.h"
        #include "vvwNet2.h"
#endif

/**
Get the current millisecond time.
*/
unsigned long __stdcall vvwGetCurMs(long lChannel);
```

```
/**
* Convert a 0..3 channel to 0, 64, 65536
* Mostly internal
* Undocumented
*/
void * __stdcall vvwChannelToHandle(long lChannel);

/**
* Convert a 0, 64, 65536 to 0..3
* Mostly internal
* Undocumented
*/
long __stdcall vvwHandleToChannel(void * hVvw);

/**
Enable or disable channels based on the bit array supplied.  VVW can contain up to 256
channels per access point.  Channels 193-255 are disabled by default.  The remaining
channels may be enabled (if the corresponding bit is set to 1) or disabled (if the
corresponding bit is set to 0) with this call.  The first 64 channels (0 through 63) are
reserved for internal ddr channels.  Then next 64 channels (64 through 127) are reserved
for VTR or DDR devices controlled via serial, Odetics or Louth protocol.  The remaining
channels are for controlling other devices through the network.  Please note that a
network channel controls all the channels on the network server box, so disabling one
network connection may disable more than one channel.  Always call GetMaxChannels()
after setting the bits to make sure all the channels you expect exist actually exist.  This
should be the first call made to the activex component.
*/
long __stdcall vvwEnableChannels(long lInternal0_31,
                                 long lInternal32_63,
                                 long lExternal64_95,
                                 long lExternal96_127,
                                 long lNetwork128_159,
                                 long lNetwork160_191);
/**
* Release memory allocated to channels
*/
long __stdcall vvwReleaseChannels(void);
/**
Returns the maximum number of channels available for control.  Channels start at 0 and
end at max channels – 1.  This return is one greater than the largest value available for
SetCurChannel(), GetCurChannel() and the lChannel parameter for the DLL interface.
*/
long __stdcall vvwGetMaxChannels(void);
/**
Returns the maximum number of internal channels available for control.  Channels start
at 0 and end at 62 – 1.  This return is one greater than the largest value available for
SetCurChannel(), GetCurChannel() and the lChannel parameter for the DLL interface.
*/
long __stdcall vvwGetMaxInternalChannels(void);
```

```
/**
Get the name of the current channel.  For unix and dll access, pass a null to get the
channel name size, then pass in a pointer that points to a memory size of at least that
many bytes (ANSI characters only).
*/
long __stdcall vvwGetChannelName(long lChannel, char * szChannelName);

/**
Returns the basic type of the channel (VTR, Internal, User, House)
        VVW_CHANTYPE_HOUSE              0x1             1
        VVW_CHANTYPE_INTERNAL           0x2             2
        VVW_CHANTYPE_VTR_DDR            0x4             4
        VVW_CHANTYPE_UNKNOWN            0xFFFFFFFF      -1
*/
long __stdcall vvwGetChannelType(long lChannel);

/**
Show the configuration dialog box for the current channel.  If the channel does not have
a configuration dialog, this function will return an error.  It is not available in Java or unix
as the dialog only shows up on the local machine, and cannot be seem through the
network.
*/
long __stdcall vvwShowConfigDialog(long lChannel, long hWnd);

//
// Transport
//

/**
Play at normal speed.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPlay(long lChannel);

/**
Play at a particular VVW speed.  VVW speeds use a base play speed of 65520.  This
means that play = 65520, reverse play = -65520, four times play = 262080, half play
speed = 32760.  Percentage play speeds may be converted to VVW speeds using the
PercentageToVVWSpeed() function. For Speed calculations please see GetSpeed() below.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPlayAtSpeed(long lChannel, long lVVWSpeed);

/**
Play from a frame to another frame.  As with editing systems, the 'from' point is included
and will be displayed but the to point is NOT included and will not be displayed.  This
means that the last frame displayed will be lFrom – 1.  The deferred flag allows
```

```
PlayFromTos to be stacked so that they will play back to back.  The deferred flag in the
status return should be false before another deferred command is added.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPlayFromTo(long lChannel, long lFrom, long lTo, BOOL fDeferred);

/**
Clip Mode Only.  Load a clip into the channel and display the lStartFrame.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwLoadClip(long lChannel, char * sz8CharClipName, long lStartFrame);

/**
Clip Mode Only.  Play the entire clip specified by clip name.  If the deferred flag is true,
clip playback will only occur once the currently playing clip has finished.  If there is no
currently playing clip, playback will occur immediately.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPlayClip(long lChannel, char * sz8CharClipName, BOOL fDeferred);

/**
Clip Mode Only.  Play the specified portion of the clip specified by clip name.  If the
deferred flag is true, clip playback will only occur once the currently playing clip has
finished.  If there is no clip currently playing, playback will occur immediately.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPlayClipFromTo(long lChannel, char * sz8CharClipName, long lFrom,
long lTo, BOOL fDeferred);

/**
Set the channel into its fastest possible forward motion state.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwFastForward(long lChannel);

/**
Set the channel into its fastest possible reverse motion state.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwFastRewind(long lChannel);

/**
Stop playback and display the current frame.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwPause(long lChannel);

/**
```

```
Seek to a particular frame and display it to the user.  This call will return before the seek
is complete.  Once the Position return in the status reaches the lFrame, the seek is
complete.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwSeek(long lChannel, long lFrame);

/**
Seek a certain number of frames from the current position.  Positive offsets imply
forward direction, negative offset imply reverse.
*/
long __stdcall vvwSeekRelative(long lChannel, long lFrameOffset);

/**
Stop the output of the controlled channel and display the input video (not supported on
all devices). On unsupported devices stop will be the same as a pause.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwStop(long lChannel);

/**
Start the channel recording. In clip mode a default clip name will be used with a duration
set to infinity. The record will stop on any transport command or at the point that the
disk is full.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwRecord(long lChannel);

/**
Record from a frame value to a frame value. As with editing systems, the 'from' point is
included and will be recorded but the to point is NOT included and will not be recorded.
This means that the last frame recorded will be lFrom – 1.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwRecordFromTo(long lChannel, long lFrom, long lTo);

/**
Clip Mode Only.  Set the clip name and length of time to record in frames.  The record
will not actually start until Record() is called.  If the lDuration is set to –1 the record will
continue until Stop() is called or the channel runs out of space.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwRecordStop(long lChannel, char * sz8CharClipName, long lDuration);

/**
Set the channels to record. Using –1 values implies that the Preset should be set to all
available channels. Record Presets will remain set until the user changes them.
        Returns 0 if successful, else an error code.
*/
```

```
long __stdcall vvwSetRecordPresets(long lChannel, long lVidEdit, long lAudEdit, long
lInfEdit);

/**
Eject the current media if it is removable.  Normally only used with VTRs.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwEject(long lChannel);

/**
* MediaCmd direct access
*/
long __stdcall vvwMediaCmd(long lChannel, void * pMediaCmd);

//
// Special Commands (lChannel must be vtr type, lTargetChannel must be internal type)
//
/**
Transfer media from one channel to another.  Only supported by VTR channels.
Currently only implemented for VTR to internal channels or internal channels to VTR
channels.  To record media from a VTR, the fToTape should be false, to record media
onto a VTR the fToTape should be true. The start and end point are from the playback
device. The edit will occur at the current timecode location on the recorder.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwTransfer(long lChannel, long lTargetChannel, long lPosition, long lStart,
long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, char * szClipName, BOOL fToTape);

/**
Retrieve the current status from the controlled device.  The status is automatically
updated by the interface, but this call ensures that the status is current when you are
checking it.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwUpdateStatus(long lChannel);

/**
Returns the current state

ctStop          0               // Stop all action
ctPause         1               // Pause, Seek
ctPlay          2               // Play at specified speed (includes pause)
ctRecord        3               // Record at specified speed
ctRecStop       4               // Stop ready for recording
ctEject         5               // Eject the current media
ctError         17              // An error has occurred
ctAbort         19              // Abort any queued commands
*/
long __stdcall vvwGetState(long lChannel);
```

```
/**
        Returns the current flags

cfDeferred = 1,                         // 0x00000001 This is a delayed
cfOverideDeferred = 1 << 30,// 0x40000000 Overide all previous deferred
commands
cfTimeMs = 1 << 1,                      // 0x00000002 Use Millisecond time for delayed
time, not fields
cfTimeTarget = 1 << 2,                  // 0x00000004 Delayed time is offset from
current time code
cfTimeHouseClock = 1 << 3,    // 0x00000008 Delayed time is based on absolute (real)
time
cfUseSpeed = 1 << 4,          // 0x00000010 Set the new speed
cfUsePresets = 1 << 5,        // 0x00000020 Use video and audio edit presets
cfUsePosition = 1 << 6,                 // 0x00000040 Use the position setting
cfUsePositionOffset = 1 << 7, // 0x00000080 Position is an offset
cfUseStart = 1 << 8,          // 0x00000100 Start a new timecode
cfUseStartOffset = 1 << 9,    // 0x00000200 Start is an offset from current tc
cfUseEnd = 1 << 10,                     // 0x00000400 End command as specified
cfUseEndOffset =  1 << 11,    // 0x00000800 End is and offset from current tc
cfUseAllIDs = 1 << 12,        // 0x00001000 Use all clip IDs
cfUseClipID = 1 << 13,        // 0x00002000 Use new clip ID, otherwise use last or
none
cfNoClipFiles = 1 << 14,      // 0x00004000 Use new clip ID, otherwise use last or
none
cfNoTCSpaces = 1 << 15,                 // 0x00008000 Use new clip ID, otherwise use
last or none
cfUseCmdAlt = 1 << 16,                  // 0x00010000 Use the dwCmdAlt
cfIsShuttle = 1 << 17,        // 0x00020000 Use speed in play for shuttle
cfFields = 1 << 20,                     // 0x00100000 Position, start and end are fields,
not frames
cfRipple = 1 << 21,                     // 0x00200000 Ripple for insert or delete
cfLoop = 1 << 22,                       // 0x00400000 Loop the clip or in out
cfTrigger = 1 << 23,          // 0x00800000 Trigger using dsync class
cfPreview = 1 << 24,          // 0x01000000 Preview set (EE, non rt play)
cfInvert = 1 << 28,                     // 0x10000000 Invert a transfer
cfTest = 1 << 29,                       // 0x20000000 See if the command exists
cfNoReturn = 1 << 31,         // 0x80000000 No return mediacmd is required
*/
long __stdcall vvwGetFlags(long lChannel);

/**
Returns the current VVW speed if the cfUseSpeed flag is set, otherwise pause or full play
speed.  VVW speeds are based on 65520 as the play speed.  To translate to decimal
number where 1.0 represents play, use the following formula:
        D1Speed = ((double)VVWSpeed / 65520.0)
For percentages, where 100.0 represents play speed, use the following formula:
        Dpercent = (((double)VVWSpeed * 100.0) / 65520.0)
```

```
                = ((double)VVWSpeed / 655.2)
XML: See \<MediaCmd\> root element, \<Speed\> sub-element

Typical VVW speeds (note speeds are linear):
Pause           0%              0
Play            100%    65520
Half Play       50%             32760
Rev Play        -100%   -65520
Rev 2 x Play-200%       131040
10 x Play       1000%   655200
Max Play        90000%          5896800
Max Rev                 -90000%-5896800
*/
long __stdcall vvwGetSpeed(long lChannel);

/**
Returns the current position if the cfUsePosition flag is set, otherwise invalid.
*/
long __stdcall vvwGetPosition(long lChannel);

/**
Returns the millisecond time the last status occurred (time of the last vertical blank).
*/
long __stdcall vvwGetLastMs(long lChannel);

/**
Returns the current start or in point if the cfUseStart flag is set.
*/
long __stdcall vvwGetStart(long lChannel);

/**
Return the current end point or out point if cfUseEnd is set.
*/
long __stdcall vvwGetEnd(long lChannel);

/**
Only supported in clip Mode. Returns the current clip name, if any.  For dll access, the
memory must be at least 9 bytes long (8 character bytes + NULL) and is always ANSI.
*/
long __stdcall vvwGetClipName(long lChannel, char * sz8CharClipName);

/**
Returns the current file name, if any.  For dll access, the memory must be at lease 261
bytes long (260 bytes max path + NULL) and is always ANSI.
*/
long __stdcall vvwGetFileName(long lChannel, char * sz260CharFileName);

/**
```

```
Returns the current time code as a string (e.g. "00:01:00:00").  For dll access, the
memory must always be at least 15 bytes long (14 byte time code plus id + NULL) and is
always ANSI.
*/
long __stdcall vvwGetCurTC(long lChannel, char * sz14ByteTC);

/**
Returns the current state as a string (e.g. "Play"). For dll access, the memory must
always be at least 15 bytes long (14 byte state + NULL) and is always ANSI.
*/
long __stdcall vvwGetCurState(long lChannel, char * sz14ByteState);

//
// Media Operation
//

// Clip Mode
/**
Clip Mode Only.  Returns the next clip identifier.  To get the first clip, szLastClip should
be an empty string.  Once the last clip available has been returned, GetNextClip will
return an error or NULL for unix/dll access.  Please note:  For unix/dll access, the
sz8CharLastClipCurClip memory area is used for the new clip.  The previous clip name is
therefore lost and the memory is not allocated by the vvw.
        Returns 0 if successful, else an error code.
*/
char * __stdcall vvwGetNextClip(long lChannel, char * sz8CharLastClipCurClip);

/**
Retuns the basic information from szClip.  The information is located in lStart, lEnd,
lVidEdit, lAudEdit and szFileName as the in point, out point, number of video channels,
number of audio channels, and the file name respectively.
        Returns 0 if successful, else an error code.
*/
long __stdcall vvwGetClipInfo (long lChannel, char * sz8CharClipName, long * lStart, long
* lEnd, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * szFileName);

/**
Retuns the extended information from szClip.  The information is located in lStart, lEnd,
lVidEdit, lAudEdit and szFileName as time of creation, last modified date, the file size,
and the number of fragments in the file respectively.
*/
long __stdcall vvwGetNextClipEx (long lChannel, char * sz8CharClipName, long *
lCreation, long * lLastModification, long * lFileSize, long * lDiskFragments);

/**
Create a virtual copy of a clip, changing the in and out points if necessary.  To use the
whole clip, set lStart to 0 and the end to –1.
        Returns 0 if successful, else an error code.
*/
```

```
long __stdcall vvwCopyClip (long lChannel, char * szSourceClip, char * szDestClip, long lStart, long lEnd);

// VTR Mode
/**
Reset the edl returns in VTR mode to the first element of the list.
*/
long __stdcall vvwEDLResetToStart(long lChannel);

/**
Returns an edit line from the VTR space of an internal channel. The function will continue to return the next edit in the timecode space until the last edit is returned, after which an error will be returned. To reset to the start of the Edl use EDLResetToStart.
        Returns 0 if successfule else an Error code.
*/
long __stdcall vvwEDLGetEdit (long lChannel, long * lRecordIn, long * lPlayIn, long * lPlayOut, long * lVidEdit, long * lAudEdit, long * lInfEdit, char * sz8CharClipName, char * sz260CharFileName);



// Shared
/**
Returns the millisecond time of the last change in the transfer queue
*/
long __stdcall vvwGetLastChangeXferMs(long lChannel);
/**
Returns the millisecond time of the last change in the current mode (clip or vtr).
*/
long __stdcall vvwGetLastChangeMs(long lChannel);

/**
*/
long __stdcall vvwInsert (long lChannel, char * szClipName, char * szFileName, long lPosition, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOL fRipple);

/**
*/
long __stdcall vvwBlank (long lChannel, char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOL fRipple);

/**
*/
long __stdcall vvwDelete (long lChannel, char * szClipName, long lStart, long lEnd, long lVidEdit, long lAudEdit, long lInfEdit, BOOL fRipple);

/**
*/
```

```
long __stdcall vvwTrim (long lChannel, long lPosition, long lStartOffset, long lEndOffset,
long lVidEdit, long lAudEdit, long lInfEdit, BOOL fRipple);

//
// Settings
//
/**
Returns the supported attributes of a get/set value  (gsClipMode, gsTcSource, etc) or –1
for not supported.
*/
long __stdcall vvwValueSupported(long lChannel, long lValueType);

/**
Returns the current setting for a get/set value.
*/
long __stdcall vvwValueGet(long lChannel, long lValueType, long * plMin, long * plMax);

/**
Sets the get/set value to setting.
*/
long __stdcall vvwValueSet(long lChannel, long lValueType, long lSetting);

/**
Sets the get/set value to setting with extended parameters.  Please set unused
parameters to NULL.
*/
long __stdcall vvwValueSet2(long lChannel, long lValueType, long lSetting, long lStart,
long lEnd, long lVidChan, long lAudChan, long lInfChan);

/**
Calls ValueXXX with gsClipMode.  If equal to 1 then the channel is in clip mode, if 0 the
channel is in VTR mode.
*/
long __stdcall vvwGetClipMode(long lChannel);
/**
Calls ValueXXX with gsClipMode.  If equal to 1 then the channel is in clip mode, if 0 the
channel is in VTR mode.
*/
long __stdcall vvwSetClipMode(long lChannel, long lSetting);

/**
Calls ValueXXX with gsTcType (drop frame, non drop frame, pal).
#TC2_TCTYPE_FILM, #TC2_TCTYPE_NDF, #TC2_TCTYPE_DF, #TC2_TCTYPE_PAL,
#TC2_TCTYPE_50, #TC2_TCTYPE_5994, #TC2_TCTYPE_60, #TC2_TCTYPE_NTSCFILM,
#TC2_TCTYPE_2398, #TC2_TCTYPE_100
*/
long __stdcall vvwGetTCType(long lChannel);
/**
Calls ValueXXX with gsTcType (drop frame, non drop frame, pal).
```

```
#TC2_TCTYPE_FILM, #TC2_TCTYPE_NDF, #TC2_TCTYPE_DF, #TC2_TCTYPE_PAL,
#TC2_TCTYPE_50, #TC2_TCTYPE_5994, #TC2_TCTYPE_60, #TC2_TCTYPE_NTSCFILM,
#TC2_TCTYPE_2398, #TC2_TCTYPE_100
*/
long __stdcall vvwSetTCType(long lChannel, long lSetting);

/**
Calls ValueXXX with gsTcSource (VITC, LTC, Control, Clip).
#GS_TCSOURCE_LTC, #GS_TCSOURCE_VITC, #GS_TCSOURCE_CTL or
#GS_TCSOURCE_CLIP
*/
long __stdcall vvwGetTCSource(long lChannel);
/**
Calls ValueXXX with gsTcSource (VITC, LTC, Control, Clip).
#GS_TCSOURCE_LTC, #GS_TCSOURCE_VITC, #GS_TCSOURCE_CTL or
#GS_TCSOURCE_CLIP
*/
long __stdcall vvwSetTCSource(long lChannel, long lSetting);

/**
Calls ValueXXX with gsAutoMode.  Required for play lists, deferred commands and auto
edit commands on VTRs.
*/
long __stdcall vvwGetAutoMode(long lChannel);
/**
Calls ValueXXX with gsAutoMode.  Required for play lists, deferred commands and auto
edit commands on VTRs.
*/
long __stdcall vvwSetAutoMode(long lChannel, long lSetting);

/**
ADD FUNCTIONS lVidEdit, lAudEdit, lInfEdit
        Returns the supported audio, video and info presets for a channel.
*/
long __stdcall vvwGetAvailablePresets(long lChannel, long * plVidEdit, long * plAudEdit,
long * plInfEdit);

/**
ADD FUNCTION lAudIn
        Get the current audio input.
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
#GS_AUDSELECT_BALANCED_10        #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
#GS_AUDSELECT_EMBEDDED
*/
long __stdcall vvwGetAudioInput(long lChannel);
/**
ADD FUNCTION lAudIn
        Set the current audio input.
```

```
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
#GS_AUDSELECT_BALANCED_10        #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
#GS_AUDSELECT_EMBEDDED
*/
long __stdcall vvwSetAudioInput(long lChannel, long lSetting);

/**
Get the current audio input level
*/
long __stdcall vvwGetAudioInputLevel(long lChannel);
/**
Get the current audio input level
*/
long __stdcall vvwSetAudioInputLevel(long lChannel, long lSetting);

/**
Get the current audio Output – See Get/SetAudioInput
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
#GS_AUDSELECT_BALANCED_10        #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
#GS_AUDSELECT_EMBEDDED
*/
long __stdcall vvwGetAudioOutput(long lChannel);
/**
Set the current audio Output – See Get/SetAudioInput
#GS_AUDSELECT_UNBALANCED_10 #GS_AUDSELECT_UNBALANCED_4
#GS_AUDSELECT_BALANCED_10        #GS_AUDSELECT_BALANCED_4
#GS_AUDSELECT_SPDIF #GS_AUDSELECT_AES_EBU
#GS_AUDSELECT_EMBEDDED
*/
long __stdcall vvwSetAudioOutput(long lChannel, long lSetting);

/**
Get the current audio output level.
*/
long __stdcall vvwGetAudioOutputLevel(long lChannel);
/**
Get the current audio output level.
*/
long __stdcall vvwSetAudioOutputLevel(long lChannel, long lSetting);

/**
Returns the RMS and Peak audio levels of the input (stop/record) or output (play/pause).
*/
long __stdcall vvwGetAudioPeakRMS(long lChannel, long lAudEdit, long * plPeaks);

/**
Get the current video input.
```

```
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
#GS_VIDSELECT_COMPONENT_YUV, #GS_VIDSELECT_COMPONENT_YUV_M2,
#GS_VIDSELECT_COMPONENT_YUV_SMPTE, #GS_VIDSELECT_COMPONENT_RGB,
#GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL, #GS_VIDSELECT_SDTI,
#GS_VIDSELECT_NONE
*/
long __stdcall vvwGetVideoInput(long lChannel);
/**
Set the current video input.
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
#GS_VIDSELECT_COMPONENT_YUV, #GS_VIDSELECT_COMPONENT_YUV_M2,
#GS_VIDSELECT_COMPONENT_YUV_SMPTE, #GS_VIDSELECT_COMPONENT_RGB,
#GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL, #GS_VIDSELECT_SDTI,
#GS_VIDSELECT_NONE
*/
long __stdcall vvwSetVideoInput(long lChannel, long lSetting);

/**
Get the current video output. See Get/SetVideoInput for settings.
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
#GS_VIDSELECT_COMPONENT_YUV, #GS_VIDSELECT_COMPONENT_YUV_M2,
#GS_VIDSELECT_COMPONENT_YUV_SMPTE, #GS_VIDSELECT_COMPONENT_RGB,
#GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL, #GS_VIDSELECT_SDTI,
#GS_VIDSELECT_NONE
*/
long __stdcall vvwGetVideoOutput(long lChannel);
/**
Set the current video output. See Get/SetVideoInput for settings.
#GS_VIDSELECT_COMPOSITE, #GS_VIDSELECT_COMPOSITE_2,
#GS_VIDSELECT_SVIDEO,
#GS_VIDSELECT_COMPONENT_YUV, #GS_VIDSELECT_COMPONENT_YUV_M2,
#GS_VIDSELECT_COMPONENT_YUV_SMPTE, #GS_VIDSELECT_COMPONENT_RGB,
#GS_VIDSELECT_D1_SERIAL, #GS_VIDSELECT_D1_PARALLEL, #GS_VIDSELECT_SDTI,
#GS_VIDSELECT_NONE
*/
long __stdcall vvwSetVideoOutput(long lChannel, long lSetting);


/**
Get the current video input's 'Setup' TBC setting.
*/
long __stdcall vvwGetVideoInputSetup(long lChannel);
/**
Set the current video input's 'Setup' TBC setting.
*/
long __stdcall vvwSetVideoInputSetup(long lChannel, long lSetting);
```

```
/**
Get the current video input's 'Video' TBC setting.
*/
long __stdcall vvwGetVideoInputVideo(long lChannel);
/**
Set the current video input's 'Video' TBC setting.
*/
long __stdcall vvwSetVideoInputVideo(long lChannel, long lSetting);
/**
Get the current video input's 'Hue' TBC setting.
*/
long __stdcall vvwGetVideoInputHue(long lChannel);
/**
Set the current video input's 'Hue' TBC setting.
*/
long __stdcall vvwSetVideoInputHue(long lChannel, long lSetting);
/**
Get the current video input's 'Chroma' TBC setting.
*/
long __stdcall vvwGetVideoInputChroma(long lChannel);
/**
Set the current video input's 'Chroma' TBC setting.
*/
long __stdcall vvwSetVideoInputChroma(long lChannel, long lSetting);

/**
Get the current global TBC's 'Setup' setting.
*/
long __stdcall vvwGetVideoTBCSetup(long lChannel);
/**
Set the current global TBC's 'Setup' setting.
*/
long __stdcall vvwSetVideoTBCSetup(long lChannel, long lSetting);
/**
Get the current global TBC's 'Video' setting.
*/
long __stdcall vvwGetVideoTBCVideo(long lChannel);
/**
Set the current global TBC's 'Video' setting.
*/
long __stdcall vvwSetVideoTBCVideo(long lChannel, long lSetting);
/**
Get the current global TBC's 'Hue' setting.
*/
long __stdcall vvwGetVideoTBCHue(long lChannel);
/**
Set the current global TBC's 'Hue' setting.
*/
long __stdcall vvwSetVideoTBCHue(long lChannel, long lSetting);
```

```
/**
Get the current global TBC's 'Chroma' setting.
*/
long __stdcall vvwGetVideoTBCChroma(long lChannel);
/**
Set the current global TBC's 'Chroma' setting.
*/
long __stdcall vvwSetVideoTBCChroma(long lChannel, long lSetting);

/**
Turn the house/reference lock on or off
*/
long __stdcall vvwGetVideoGenlock(long lChannel);
/**
Turn the house/reference lock on or off
*/
long __stdcall vvwSetVideoGenlock(long lChannel, long lSetting);

/**
Set the genlock source to input or external reference
*/
long __stdcall vvwGetVideoGenlockSource(long lChannel);
/**
Set the genlock source to input or external reference
*/
long __stdcall vvwSetVideoGenlockSource(long lChannel, long lSetting);

/**
Get the current compression rate
*/
long __stdcall vvwGetCompressionRate(long lChannel);
/**
Set the current compression rate
*/
long __stdcall vvwSetCompressionRate(long lChannel, long lSetting);

/**
Get the status of the super imposed time code overlay
*/
long __stdcall vvwGetSuperImpose(long lChannel);
/**
Set the status of the super imposed time code overlay
*/
long __stdcall vvwSetSuperImpose(long lChannel, long lSetting);
/**
Get the ms time the last error was added to the error log
*/
long __stdcall vvwGetErrorLogMs();
/**
```

```
Set the error log pointer to the message you want
*/
long __stdcall vvwSetErrorLog(long lSetting);
/**
Get the number of current errors
*/
long __stdcall vvwGetNumberOfErrors(long * plErrors);
/**
Get the length of the current error string
*/
long __stdcall vvwGetErrorLength(long * plLastError, long * plErrorLength);
/**
Get the current error.  Sets pointer to the next one automatically
*/
long __stdcall vvwGetError(long * plLastError, long * plSeverity, char * szError, long *
lTime);

/**
Returns the total number of frames of storage available at current compression rate if
the storage space was empty.
*/
long __stdcall vvwGetTotalTime(long lChannel);
/**
Returns the remaining number of frames of storage available at current compression
rate.
*/
long __stdcall vvwGetFreeTime(long lChannel);
/**
Returns the total storage connected in megabytes.
*/
long __stdcall vvwGetTotalStorage(long lChannel);
/**
Returns the amount of available storage for recording in megabytes.
*/
long __stdcall vvwGetFreeStorage(long lChannel);

/**
Get the available commands for a channel.
*/
long __stdcall vvwGetChannelCapabilities(long lChannel);

/**
Returns the version string of the VVW subsystem.
*/
char * __stdcall vvwGetVVWVersion();

/**
Returns the version string of the MediaReactor subsystem.
*/
```

```
char * __stdcall vvwGetMRVersion();

/**
Returns the type string of the VVW channel.
*/
char * __stdcall vvwGetVVWType(long lChannel);

/**
Get the name of a picon file from the media file's name
*/
long __stdcall vvwGetPiconName(char * szFileName);

/**
Start playback at a specified MS
*/
long __stdcall vvwPlayAtMs(long lChannel, long lMs);

/**
Start Recording at a specified MS
*/
long __stdcall vvwRecordAtMs(long lChannel, long lMs);

//
// Utility
//
/**
Free a string value returned by the channel.
*/
void __stdcall vvwFreeString(char * szString);


#if !defined(_VVW_TYPES_HAVE_ALREADY_BEEN_INCLUDED)

// Various speed limits and definitions
//! Forward play speed (normal) in VVW (65520) see MEDIACMD::lSpeed
#define SPD_FWD_PLAY                    65520L
//! Pause speed (0%) in VVW (0) see MEDIACMD::lSpeed
#define SPD_PAUSE                       0L
//! Reverse play speed (-100%) in VVW (-65520) see MEDIACMD::lSpeed
#define SPD_REV_PLAY                    (-SPD_FWD_PLAY)
//! Maximum possible play speed in VVW see MEDIACMD::lSpeed
#define SPD_FWD_MAX                             5896800
//! Minimum possible play speed in VVW see MEDIACMD::lSpeed
#define SPD_REV_MAX                     (-SPD_FWD_MAX)
//! Illegal speed, set MEDIACMD::lSpeed to this value if not used
#define SPD_ILLEGAL                     2147483647L

#endif
```

Appendix IV – MediaCmdNet.h

Now available for Linux and Win32 as share library/DLL.