

# VW Control Specification

©opyright 2004, Drastic Technologies Ltd

Drastic Technologies

12 Drummond Street, Unit 3

Toronto, ON, M8V 1Y8

CANDADA

(416) 255 5636

(Fax) 255 8780

<http://www.drastictech.com>

# Contents

<b>VVW CONTROL SPECIFICATION.....</b>	<b>1</b>
<b>CONTENTS.....</b>	<b>2</b>
<b>INTRODUCTION.....</b>	<b>4</b>
<b>REFERENCES.....</b>	<b>5</b>
<b>INTERFACE.....</b>	<b>6</b>
<b>Basic Information.....</b>	<b>6</b>
GetVVWVersion.....	7
GetMRVersion.....	7
GetVVWType.....	7
<b>Configuration.....</b>	<b>8</b>
ShowConfigDialog.....	8
<b>Utility Functions.....</b>	<b>9</b>
GetCurMs.....	9
TCMaxFrame.....	9
TCToFrame.....	9
TCToString.....	10
VVWSpeedToPercentage.....	10
PercentageToVVWSpeed.....	10
<b>Command Interface.....</b>	<b>12</b>
MediaCmd - Method.....	12
MediaCmd - Event.....	14
Channel.....	15
Cmd.....	15
Flags.....	15
Speed.....	15
VideoChannels.....	15
AudioChannels.....	15
InfoChannels.....	15
CmdAltMs.....	15
Position.....	15
Start.....	15
End.....	15
ID.....	15
ClipID.....	15

FileID.....	15
<b>Current Status Properties.....</b>	<b>16</b>
CurCmd.....	16
CurFlags.....	16
CurSpeed.....	17
CurVideoChannels.....	17
CurAudioChannels.....	17
CurInfoChannels.....	18
CurCmdAltMs.....	18
CurPosition.....	18
CurStart.....	18
CurEnd.....	18
CurID.....	18
CurClipID.....	18
CurFileID.....	19
<b>Typical Commands From Controller.....</b>	<b>20</b>
Transport.....	20
CtPlay.....	20
CtPause.....	20
CtStop.....	20
CtRecStop.....	21
CtRecord.....	21
CtEject.....	21
CtTransfer.....	21
CtInsert.....	21
CtBlank.....	21
ctDelete .....	21
CtTrim.....	21
CtChanSelect.....	21
CtGetState.....	22
CtSetState.....	22
CtGetValue.....	22
CtSetValue.....	22
CtValueSupported.....	22
CtError.....	22
CtTerminate.....	22
CtAbort.....	22
Status Returns.....	23
Setup And Settings.....	24
 <b>APPENDIX I – MEDIACMD.H.....</b>	 <b>25</b>
 <b>APPENDIX II – MEDIACMD.BAS.....</b>	 <b>108</b>

## Introduction

The VWW interfaces are designed to allow VWW customers, OEMs and Drastic component OEMs to create custom control solutions in the simplest manner possible. A good background document for understanding this SDK is the 'VWW Interface Specification'. That document describes some of the clients that have been developed for this SDK component.

The VWW Control Component uses an ActiveX abstraction of the MediaCmd interface layer to provide remote control of your application including these protocols:

- Sony RS-422 Serial 9 Pin VTR
- Sony D2/DigiBeta/HD Serial Protocol Extensions (selected)
- Panasonic MII/DVCPPro Serial Protocol Extensions (selected)
- JVC D9 Protocol Extensions (selected)
- Odetics Server Protocol
- Louth Server Protocol
- VDCP Server Protocol
- HP Server Protocol
- Tektronics/Grass Valley Odetics/Louth Extensions (selected)

All the above protocols are distilled into a single command set, allowing the application to respond to all supported protocols with the same command and response structure. The control component is still under active development, and new may be added seamlessly to application supporting this interface.

For speed and stability, the control component is made up of two major parts:

DComCtrl.sys – System level VTR/DDR emulator/communications handler

VvwCTL.dll – User level interface to VTR DDR emulator

This division completely separates the application from the real time (9 ms) requirements of the serial device. The application may be written in any language that supports ActiveX, as the system driver will make sure that the application does not interfere with the control. Due to this separation, it is critical that the system driver is aware of the timing of the vertical blank. Status/Position returns must be stamped with the actual blank time the frame occurred to ensure frame accurate control. The other required components are:

ClipCtrl.dll – Basic clip database manipulation (internal)

Dlist.dll – Shared doubly linked list (internal)

Dsync.dll – Timing, house sync (available through activex)

ErrMsg.dll – Standard error output (internal)

TCXlat.dll – Time code/frame conversion (available through activex)

VvUtil.dll – Utility support functions (internal)

## References

- MediaCmd.h - Internal media cmd structures on which this interface is based.
- MediaCmd.bas - Visual Basic version of MediaCmd.h
- vwwCtl.tlb - MediaControlX type library
  
- MediaCtrl.chm - Interface detail help file from headers (Doxygen generated)

# Interface

The following is assumed:

Language/Platform	Name	Notes
ActiveX VB	Vbx	ActiveX Control
ActiveX C++	Pcx	Pointer to ActiveX Control

- All pointers for ActiveX C++, DLL and Unix must be valid and correctly sized
- Any BSTR returns are freed by the caller using the function SysFreeString()
- ClipID has a maximum of 8 alpha numeric characters as per the Louth and Odetics specifications
- FileID is DOS/Windows formatted in one of two forms
  - X:\Some Path\On The Drive\Media Files Name.ext
  - \\VWWSERVER\X\ Some Path\On The Drive\Media Files Name.ext

Nomenclature:

## ClipID

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] = "\0";           // Include unused 1 char safety
Char pClipName = new char[9];
Char szpClipName = malloc(9);
```

## FileID

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] = "\0";        // 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] = "\0";        // Include unused 1 char safety
Char pClipName = new char[15];
Char szpClipName = malloc(15);
```

## Basic Information

### **GetVWVersion**

ActiveX VB	String vbx.GetVWVersion()
ActiveX C++	BSTR pcx->GetVWVersion()

Returns a string containing the version of the VW control interface.

### **GetMRVersion**

ActiveX VB	String vbx.GetMRVersion()
ActiveX C++	BSTR pcx->GetMRVersion()

Returns a string containing the version of the MediaReactor Core control interface.

### **GetVWType**

ActiveX VB	String vbx.GetVWType()
ActiveX C++	BSTR pcx->GetVWType()

Returns a string containing the version of the VW control type.

## Configuration

### ShowConfigDialog

ActiveX VB	vb.ShowConfigDialog (0) As Long
ActiveX C++	long pcx->ShoConfigDialog (hWnd)

This displays the control setup dialog to the user. It allows for enabling and disabling of protocols, changing of communications interface and timing setup. The call does not return until the user closes the dialog.

Returns 0 or an error code if user cancels.

## Utility Functions

### GetCurMs

ActiveX VB	vb. GetCurMs () As Long
ActiveX C++	long pcx-> GetCurMs ()

Returns the current value of the performance clock in milliseconds. This is the clock that must be used to send vertical blank timings to the controller. The client application may create this timing with the following code:

```
static unsigned __int64 freq = {0};
// Using milli seconds
DWORD __stdcall localGetCurMs()
{
    unsigned __int64 p;
    if(freq == 0) {
        QueryPerformanceFrequency((PLARGE_INTEGER)&freq);
    }
    QueryPerformanceCounter((PLARGE_INTEGER)&p);
    return (DWORD)((p * 1000i64) / freq);
}
```

### TCToFrame

ActiveX VB	vb. TCToFrame (IFlags As Long) As Long
ActiveX C++	long pcx-> TCToFrame (long IFlags)

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

### TCToFrame

ActiveX VB	vb. TCToFrame (szTC As String, IFlags As Long) As Long
ActiveX C++	Long pcx-> TCToFrame (BSTR szTC, long IFlags)

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	vb. TCToString (ITC As Long, IFlags As Long) As String
ActiveX C++	BSTR pcx-> TCToString (long ITC, long IFlags)

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

### VVWSpeedToPercentage

ActiveX VB	vb. VVWSpeedToPercentage (IVVWSpeed As Long) As Double
ActiveX C++	Double pcx-> VVWSpeedToPercentage (long IVVWSpeed)

Convert a VVW speed (65520 based) to a percentage based speed (100.0). See CurSpeed() property for more information.

### PercentageToVVWSpeed

ActiveX VB	vb. PercentageToVVWSpeed (double ddPercentageSpeed) As Long
------------	---

ActiveX C++

Long pcx-> PercentageToVWWSpeed (double  
ddPercentageSpeed)

Convert a percentage speed (100.0) to a VW speed (65520). See CurSpeed()  
property for more information.

## Command Interface

### MediaCmd - Method

ActiveX VB	vbv.MediaCmd (ByRef ctCmd As Long, ByRef cfFlags As Long, ByRef lSpeed As Long, ByRef dwVideoChannels As Long, ByRef dwAudioChannels As Long, ByRef dwInfoChannels As Long, ByRef dwCmdAltMs As Long, ByRef dwPosition As Long, ByRef dwStart As Long, ByRef dwEnd As Long, ByRef ID As String, ByRef dwChannels As Long) As Long
ActiveX C++	Long pcv->MediaCmd (long * ctCmd, long * cfFlags, long * lSpeed, long * dwVideoChannels, long * dwAudioChannels, long * dwInfoChannels, long * dwCmdAltMs, long * dwPosition, long * dwStart, long * dwEnd, BSTR * ID, long * dwChannels)

Send a MediaCmd to the controller. This is generally used to change settings such as time code type, time code source, video standard, etc. In general, most status changes are read by a status request through the MediaCmd(Event).

#### Parameters:

CtCmd	Main Command Type: CtStop, ctPause, ctPlay, ctRecord, ctRecStop, ctEject, ctTransfer, ctInsert, ctBlank, ctDelete, ctTrim, ctChanSelect, ctGetState, ctSetState, ctGetValue, ctSetValue, ctValueSupported, ctError, ctTerminate, ctAbort
CfFlags	CfDeferred, cfOverrideDeferred, cfTimeMs, cfTimeTarget, cfTimeHouseClock, cfUseSpeed, cfUsePresets, cfUsePosition, cfUsePositionOffset, cfUseStart, cfUseStartOffset, cfUseEnd, cfUseEndOffset, cfUseAllIDs, cfUseClipID, cfNoClipFiles, cfNoTCSpaces, cfUseCmdAlt, cfIsShuttle, cfFields, cfRipple, cfLoop, cfTrigger, cfPreview, cfInvert, cfTest, cfNoReturn
lSpeed	VW Speed: 0 = pause, -65520 = reverse play, 65520 = play (linear scale)
DwVideoChannels	Bit array of affected video channels (1=0, 2=1, 4=2, 8=3)
DwAudioChannels	Bit array of affected audio channels (1=0, 2=1, 4=2, 8=3)
DwInfoChannels	Bit array of affected information channels (1=0, 2=1, 4=2, 8=3)
DwCmdAltMs	Either a gsXXX command for Set/GetValue/Supported or the vertical blank timing for position frame or target (depends on flags)
DwPosition	The current frame position, the target frame position or a position offset
DwStart	The current frame start, the target frame start or a start offset (inclusive)
DwEnd	The current frame end + 1, the target frame end + 1 or a end offset (exclusive)
ID	ID string for storing ClipID, FileID or other textual information
DwChannel	Target channel for command (0..255)

For detailed information on these parameters, please see the MediaCmd Parameters below. Returns 0 or an error code

## MediaCmd - Event

ActiveX VB	Vbx_MediaCmd (ctCmd As Long, cfFlags As Long, lSpeed As Long, dwVideoChannels As Long, dwAudioChannels As Long, dwInfoChannels As Long, dwCmdAltMs As Long, dwPosition As Long, dwStart As Long, dwEnd As Long, ID As String, dwChannels As Long) As Long
ActiveX C++	Long pcx->MediaCmd (long ctCmd, long cfFlags, long lSpeed, long dwVideoChannels, long dwAudioChannels, long dwInfoChannels, long dwCmdAltMs, long dwPosition, long dwStart, long dwEnd, BSTR ID, long dwChannels)

This event is sent by the controller to the application whenever a transport command is received, a status is required or other information is required. All commands should be handled in some way, but the most important are:

- ctGetState – Status request, return current cmd, speed, position and vertical blank time
- ctGetValue – gsTcType, gsSignalFormat, gsFirstClip, gsNextClip, gsClipInfo, gsPlayDelay
- ct(Transport) – Play, Pause, Stop, Record, etc

### Parameters:

CtCmd	Main Command Type: CtStop, ctPause, ctPlay, ctRecord, ctRecStop, ctEject, ctTransfer, ctInsert, ctBlank, ctDelete, ctTrim, ctChanSelect, ctGetState, ctSetState, ctGetValue, ctSetValue, ctValueSupported, ctError, ctTerminate, ctAbort
CfFlags	CfDeferred, cfOverrideDeferred, cfTimeMs, cfTimeTarget, cfTimeHouseClock, cfUseSpeed, cfUsePresets, cfUsePosition, cfUsePositionOffset, cfUseStart, cfUseStartOffset, cfUseEnd, cfUseEndOffset, cfUseAllIDs, cfUseClipID, cfNoClipFiles, cfNoTCSpaces, cfUseCmdAlt, cfIsShuttle, cfFields, cfRipple, cfLoop, cfTrigger, cfPreview, cfInvert, cfTest, cfNoReturn
lSpeed	VW Speed: 0 = pause, -65520 = reverse play, 65520 = play (linear scale)
DwVideoChannels	Bit array of affected video channels (1=0, 2=1, 4=2, 8=3)
DwAudioChannels	Bit array of affected audio channels (1=0, 2=1, 4=2, 8=3)
DwInfoChannels	Bit array of affected information channels (1=0, 2=1, 4=2, 8=3)
DwCmdAltMs	Either a gsXXX command for Set/GetValue/Supported or the vertical blank timing for position frame or target (depends on flags)
DwPosition	The current frame position, the target frame position or a position offset
DwStart	The current frame start, the target frame start or a start offset (inclusive)
DwEnd	The current frame end + 1, the target frame end + 1 or a end offset (exclusive)
ID	ID string for storing ClipID, FileID or other textual information
DwChannel	Target channel for command (0..255)

With each of these commands, returns may be set through the following properties:

**Channel**  
Normally zero, minus one. Leave at default.

**Cmd**  
Current command state (ctPlay, ctPause, ctRecord, etc)

**Flags**  
Valid data and modifier flags (cfUsePosition, cfDeferred, cfTimeMs, etc)

**Speed**  
Current VVW Speed (see CurSpeed() for more details)

**VideoChannels**  
Current video channels (bit array)

**AudioChannels**  
Current audio channels (bit array)

**InfoChannels**  
Current information channels (bit array)

**CmdAltMs**  
Current vertical blank time

**Position**  
Current position at vertical blank time

**Start**  
Current start point (inclusive)

**End**  
Current end point (exclusive)

**ID**  
Current ID string. Normally made up of ClipID and FileID.

**ClipID**  
Current 8 character odetics/louth clip name

**FileID**  
Current file name of 8 character odetics/louth clip name



- 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

## CurSpeed

ActiveX VB	vbv. CurSpeed () As Long
ActiveX C++	long pcx-> CurSpeed ()

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:

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

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

## CurVideoChannels

ActiveX VB	vbv. CurVideoChannels () As Long
ActiveX C++	long pcx-> CurVideoChannels ()

Returns the current channel bit array if the cfUsePresets flag is set, otherwise invalid.

## CurAudioChannels

ActiveX VB	vbv. CurAudioChannels () As Long
ActiveX C++	long pcx-> CurAudioChannels ()

Returns the current channel bit array if the cfUsePresets flag is set, otherwise invalid.

## CurInfoChannels

ActiveX VB	vb. CurInfoChannels () As Long
ActiveX C++	long pcx-> CurInfoChannels ()

Returns the current channel bit array if the cfUsePresets flag is set, otherwise invalid.

## CurCmdAltMs

ActiveX VB	vb. CurCmdAltMs () As Long
ActiveX C++	long pcx-> CurCmdAltMs ()

Returns the current vertical blank time in milliseconds if the cfUseCmdAlt and cfTimeMs flags are set, otherwise invalid.

## CurPosition

ActiveX VB	vb. CurPosition () As Long
ActiveX C++	Long pcx-> CurPosition ()

Returns the current position if the cfUsePosition flag is set, otherwise invalid.

## CurStart

ActiveX VB	vb. CurStart () As Long
ActiveX C++	long pcx-> CurStart ()

Returns the current start or in point if the cfUseStart flag is set.

## CurEnd

ActiveX VB	vb. CurEnd () As Long
ActiveX C++	long pcx-> CurEnd ()

Return the current end point or out point if cfUseEnd is set.

## CurID

ActiveX VB	vb. CurID () As String
ActiveX C++	BSTR pcx-> CurID ()

Return the current ID field, normally made up of ClipID and FileID.

## CurClipID

ActiveX VB	vb. CurClipID () As String
ActiveX C++	BSTR pcx-> CurClipID ()

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.

## CurFileID

ActiveX VB	Vbx. CurFileID () As String
ActiveX C++	BSTR pcx-> CurFileID ()

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

## Typical Commands From Controller

### Transport

#### CtPlay

Play commands cause the client to begin playing audio and video. Often the play will be modified by a speed. If the cfSpeed flag is set, the lSpeed will contain the required speed. There are two types of speed command. If the cfIsShuttle flag is set, then the video/audio output need not be perfect. This is typically a jog or shuttle command. If it is not set, then the output should be of the highest quality possible. This is typically a normal play, var or dmc (dynamic motion control) command.

Play may also be modified with a clip name under clip based protocols. If a clip name is supplied in ClipID, then the cfUseClipID flag will be set.

A play may also include a first frame position (if cfUsePosition is set). A start and end may also be specified in Start and End if the cfUseStart and/or cfUseEnd flags are set. Please note that the end point is always exclusive of the edit or playback. This means the frame specified by the end is never displayed or recorded on.

The cfDeferred flag may also be set. This is used for non linear, seamless clip or section playback. Please see the section on list playback for more information.

Standard Play	No Flags
Standard Play	CfUseSpeed, Speed == 65520
Shuttle Play	CfUseSpeed + cfIsShuttle
DMC Play	CfUseSpeed
Play Clip	CfUseClipID
Play From	CfUseStart or cfUsePosition
Play To	CfUseEnd
Play From To	CfUseStart + cfUseEnd
Play at specific time (may incl above)	CfUseCmdAlt + cfTimeMs = cfTimeTarget

#### CtPause

Pause is used to display the current frame, or seek and display a frame.

Pause at current frame	No Flags
Seek to frame	CfUsePosition
Prepare Next Clip	CfUsePostition + cfUseClipID + cfDeferred
Prepare Next Clip Start End	CfUsePostition + cfUseClipID + cfDeferred + cfUseStart + cfUseEnd

#### CtStop

Stop playback and display passthrough audio/video if possible. If not, then go to pause state. Stop may include a position.

Stop at current frame	No Flags
Stop and set frame	CfUsePosition

## CtRecStop

Stop playback and display passthrough audio/video. Setup for the next clip record. This command will always include a clip name and will only be received before a ctRecord from clip based protocols. Standard Sony protocol simply sends the record.

Prepare to record	CfUseClipID
Prepare to record from	CfUseClipID + (cfUsePosition or cfUseStart)
Prepare to record from to	CfUseClipID + cfUseStart + cfUseEnd

## CtRecord

Begin recording from video/audio input to disk. In server mode this will be preceded by ctRecStop. In VTR mode, there is no 'prepare record' command. The basic record types are record a clip, crash record, assemble record and insert record.

Record a clip	(preceded by ctRecStop)
Crash record	No Flags
Assemble Record	CfUseStart + cfTimeMs + cfTimeTarget
Insert Record	CfUseStart + cfTimeMs + cfTimeTarget + cfUsePresets

Once a record is initiated, there are a number of ways it may end.

CtStop - Stop record as quickly as possible

CtPause - Stop record and go to pause mode on next frame after record

CtPlay - Go from record to play. Normally this command has cfUseEnd flag specifying the last frame to record + 1. Playback will continue on the specified end when it is reached.

## CtEject

Eject the current media. Ignored by most devices.

## CtTransfer

Internal command. Not sent by controller.

## CtInsert

Internal command. Not sent by controller.

## CtBlank

Internal command. Not sent by controller.

## ctDelete

Internal command. Not sent by controller.

## CtTrim

Internal command. Not sent by controller.

## CtChanSelect

Select channels for record or edit to edit display.

Set Preset	CfUsePreset
Change edit to edit	CfUsePreset + cfTimeMs + cfTimeTarget

### CtGetState

Return the current state of the client. See 'Status Returns' section.

### CtSetState

Internal command. Not sent by controller.

### CtGetValue

### CtSetValue

### CtValueSupported

Set, get and check support for specific commands. See 'Setup and Settings' section.

### CtError

An error has occurred.

### CtTerminate

The controller is terminating.

### CtAbort

The command should be aborted. Go to pause from play and stop from record.

## **Status Returns**

Status returns are critical to the correct operation of the controller.

## Setup And Settings

## Appendix I – MediaCmd.h

```
/******  
* $Header: /Ass/inc/MediaCmd.h 92 5/25/01 10:13p Administrator $  
*  
* $Workfile: MediaCmd.h $  
* $Author: Administrator $  
* $Revision: 92 $  
* $Archive: /Ass/inc/MediaCmd.h $  
*  
* Copyright (C) 1998 Drastic Technologies Ltd. All Rights Reserved.  
* 12 Drummond St. Unit 3, Toronto, ON, M8V 1Y8 (416) 255 5636  
* engineering@drastictech.com http://www.drastictech.com  
*  
*****/  
  
//  
// Common header describing the command interface between  
// media control modules.  
//  
#include <dlist.h>  
  
#ifndef _MEDIACMD_INCLUDED_H  
#define _MEDIACMD_INCLUDED_H  
  
// 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::ISpeed  
#define SPD_FWD_PLAY 65520L  
//! Pause speed (0%) in VVW (0) see MEDIACMD::ISpeed
```

```

#define SPD_PAUSE                0L
//! Reverse play speed (-100%) in VVW (-65520) see MEDIACMD::ISpeed
#define SPD_REV_PLAY             (-SPD_FWD_PLAY)
//! Maximum possible play speed in VVW see MEDIACMD::ISpeed
#define SPD_FWD_MAX              5896800
//! Minimum possible play speed in VVW see MEDIACMD::ISpeed
#define SPD_REV_MAX              (-SPD_FWD_MAX)
//! Illegal speed, set MEDIACMD::ISpeed 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::ISpeed) 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::cfOverrideDeferred) 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 receives
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
#GS_CLIP_INFO
* 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
* <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 position (+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.
* <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 (vwNet.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 (vwInt.h, vwExt.h,
vwNet.h-as controller,
* vwDS2.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 seamlessly back to back.
        * <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>
*/
cfOverrideDeferred = 1 << 30, // 0x40000000 Override 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 default 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 occurrence. 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 VWV 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
/**
* Means the command should not be used on any clip or clip spaces
* <HR>
*/
cfNoClipFiles = 1 << 14,      // 0x00004000 Use new clip ID, otherwise use
last or none
/**
* Means the command should not be used on any clip within or the TCspace
itself
* <HR>
*/
cfNoTCspaces = 1 << 15,        // 0x00008000 Use new clip ID,
otherwise use last or none
/**
* 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
*/
cfIsShuttle = 1 << 17,        // 0x00020000 Use speed in play for shuttle
/**
* 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)
/**
* 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 cfOverrideDeferred
    /**
    * 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 = 1 << 31,           // 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
= 1 << 31,
        vidChanAll = 0xFFFFFFFF
};
/*! @}
```

```
/*! 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 = 1 << 31,
        audChanAll = 0xFFFFFFFF
};
/*! @}
```

```
/*! 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 information
        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 = 1 << 31,
    infChanAll = 0xFFFFFFFF
};
//! @{

/**
 * Enum sent in MEDIACMD::dwCmdAlt for the commands cmdType::ctGetValue,
 * cmdType::ctSetValue, cmdType::ctValueSupported.
 */
enum cmdGetSetValue {
    /**
     * Current internal time - control or clip absolute
     * \li cmdType::ctSetValue
     * <BR> nada
     * \li cmdType::ctGetValue
     * <BR> MEDIACMD::dwPosition
     * <HR>
     */
    gsTc = 1, // Current internal TC (dwPosition)
    /**
     * Current internal user bits - control or clip absolute
     * <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
     * \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
* \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
* <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)

```

```

/**
 * Last known 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)
/**
 * Last known 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 Pioneer dual head emulation. For Louth and Odetics
* enables preview play look ahead for seamless 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 receiving 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

/**
* Return the next clip in the current clip space. If the previous clip
* 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 clip id or 8 NULLs for first clip
* <BR> -out- MEDIACMD::arbID - Next 8 character id or 8 NULLs if clip list
complete
* <BR> for MCMD2 -out- MEDIACMD::arbID - Next 8 character id and unc file
path separated by NULL or 8 NULLs if clip list complete
* <BR> MEDIACMD::dwPosition - Current position in the clip
* <BR> MEDIACMD::dwStart - First frame of clip
* <BR> MEDIACMD::dwEnd - Last frame of clip
* <HR>
*/
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      0      0
300             1      2      0      file1  - 10 sec
VA2 from file1
* gsTCSGetTLNextState 0      0      0      0
*                   0      0      0      - First state 0
*   Rtn          0      0      0      0
*                   1      2      0      0
- First clip channels
*   ( Copy prev gsTCSGetTLNextState into gsTCSGetTLClipInfo before
sending )
* gsTCSGetTLClipInfo  0      1
*                   2      0      0      - Last get state
*   Rtn          0      0      0      0
150             0      1      0      file2  - 5 sec
A1 from file2
*   ( Use last gsTCSGetTLNextState for this call )
* gsTCSGetTLNextState 0      1
*                   2      0      0      - Use last state to get
next
*   Rtn          0      0
*                   1      3      0      0
- Channels used so far
*   ( Copy prev gsTCSGetTLNextState into gsTCSGetTLClipInfo before
sending )
* gsTCSGetTLClipInfo  0      1
*                   3      0      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      0      - Use last state to get
next

```

```

*      Rtn          0          1          150
*      0          0          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
immediatly 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::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)

// 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 VWV 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
* <BR>
* \li cmdType::ctSetValue
* <BR>          - Not Supported
* \li cmdType::ctGetValue
* <BR>          -in- MEDIACMD::dwAudioChannels - Requested channels to
check
* <BR>          MEDIACMD::dwStart - HIWORD=Peak channel 0,
LOWORD=RMS channel 0 (range 0-65535)
* <BR>          MEDIACMD::dwEnd - HIWORD=Peak channel 1, LOWORD=RMS
channel 1 (range 0-65535)
* <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)
/**
* 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>
*/
gsAudInputSampleRate,
/**
* 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>
*/
gsAudInputMode,
/**
* 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>
*/
gsAudInputHeadRoom,
/**
* 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>
*/
gsAudInputOriginal,
/**

```

```

* 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>
*/
gsAudInputErrorProtect,
/**
* 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>
*/
gsAudInputCopyright,
/**
* 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>
*/
gsAudInputSlave,
/**
* 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 pre-read 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 passthrough video
        * \li cmdType::ctGetValue
        * <BR>          MEDIACMD::dwPosition - #GS_TRUE Always in playback mode,
else if #GS_FALSE then will passthrough 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)

```

```

// 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
(0x0000-0xffff0)
* <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 freeze state
* <BR>
* \li cmdType::ctSetValue
* <BR>         MEDIACMD::dwPosition - 0 normal, !0 frozen
* \li cmdType::ctGetValue
* <BR>         MEDIACMD::dwPosition - 0 normal, !0 frozen
* <HR>
*/
gsFreeze,
/**
* 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
(0x0000-0xffff0)
* <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, //
/**
*         Size of picture
*/
gsMpegVerticalRes = 700,
/**
*         Size of picture
*/
gsMpegHorizontalRes,
/**
*         4:0:0 -- 4:2:0 -- 4:2:2
*/
gsMpegChromaFormat,
/**
*
*/
gsMpegDCPrecision,
/**
*
*/
gsMpegAspectRatio,
/**
*
*/
gsMpegStandard,
/**
*Video/Audio Language Code
*/
gsMpegLanguageCode,
/**
*         Closed Captioning Format
*/
gsMpegCCFormat,
/**
*
*/
gsMpegConcealmentVector,
/**
*
*/

```

```
gsMpegClosedGop,  
/**  
*  
*/  
gsMpegAdjustGopTC,  
/**  
*  
*/  
gsMpegAltCoEffTable,  
/**  
*  
*/  
gsMpegNonLinearQuant,  
/**  
*  
*/  
gsMpegMuxRate,  
/**  
*  
*/  
gsMpegAudPacketSize,  
/**  
*  
*/  
gsMpegVidPacketSize,  
/**  
*  
*/  
gsMpegAudioStreamID,  
/**  
*  
*/  
gsMpegVideoStreamID,  
/**  
*  
*/  
gsMpegAudioStreamPID,  
/**  
*  
*/  
gsMpegVideoStreamPID,  
/**  
* 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
 * <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_UN8BIT 8Bit Y'CrCb uncompressed video
 * <BR>#GS_COMPTYPE_UN10BIT 10Bit Y'CrCb uncompressed video
 * <BR>#GS_COMPTYPE_HDPAN Panasonic HD to SDI codec
 * <BR>#GS_COMPTYPE_HDSOXY Sony HD to SDI codec
 * <BR>#GS_COMPTYPE_HDUNCOMP Uncompressed Y'CrCb High Def Video
 * \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_UN8BIT 8Bit Y'CrCb uncompressed video
* <BR>#GS_COMPTYPE_UN10BIT 10Bit Y'CrCb uncompressed video
* <BR>#GS_COMPTYPE_HDPAN Panasonic HD to SDI codec
* <BR>#GS_COMPTYPE_HDSOXY Sony HD to SDI codec
* <BR>#GS_COMPTYPE_HDUNCOMP Uncompressed Y'CrCb High Def Video
* <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
/**
* 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>
*/
gsCompIFactor,          // 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>
*/
gsCompBFactor,          // 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>
*/
gsCompPFactor,         // 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>
*/
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)
/**
* Select recording drive
* <BR>
* \li cmdType::ctGetValue
* <BR>          MEDIACMD::dwPosition - gsTrue/gFalse
* <BR>          MEDIACMD::arbID - new file name form LTC Generator / 422
Device / Internal Clock
* <BR> MEDIACMD::cfFlags - must be set to cfUseClipID
                                and could use cfUseEnd

for Drive Time Remaining
* <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 supfiletypes
(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/Inhibits 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/Inhibits 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 VVW version number
* <BR>
* \li cmdType::ctSetValue
* <BR>          Not Supported
* \li cmdType::ctGetValue

```

```

* <BR>          MEDIACMD::arbID - Zero terminated ansi string with version
number
* <HR>
*/
gsVWVVersion = 60000,          // VWV 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 VWV type description
* <BR>
* \li cmdType::ctSetValue
* <BR>          Not Supported
* \li cmdType::ctGetValue
* <BR>          MEDIACMD::arbID - Zero terminated ansi string with VWV
machine type
* <HR>
*/
gsVWVType,                    // Name of vvw model (dwPosition,
arbID = string)
/**
* Get VWV channel type
* <BR>
* \li cmdType::ctSetValue
* <BR>          Not Supported
* \li cmdType::ctGetValue
* <BR>          MEDIACMD::arbID - Zero terminated ansi string with channel
type
* <HR>
*/
gsVWVChannelType,           // Name of channel (dwPosition, arbID = string)

/**
* Get VWV version number
* <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

/**
* Get VVW version number
* <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>
*/
gsHwnds,                  // Get above info

/**
* 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 - if 1 then clip mode, else time code
mode
* \li cmdType::ctGetValue

```

```

* <BR>          MEDIACMD::dwPosition - if 1 then clip mode, else time code
mode
* <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 I-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 I-X (16-23/ 17-24)
 * <BR>     MEDIACMD::dwPosition - last change in ms aligned with Dsync
 * <HR>
 */
gsGPIIn,                                     // Get/Set GPI inputs

/**
 * Get last change millisecond time from clip space, tc space or file
 * <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 I-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 I-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

/**
 * Save the current parameters to the registry
 * <BR>
 * \li cmdType::ctSetValue
 * <BR>     Not Used
 * \li cmdType::ctGetValue
 * <BR>     Not Used
 * <HR>
 */
gsSaveCurrent = 100000,                      // Save the current params
};

// 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

//! Audio in/out unbalanced (RCA connector) high impedance at -10db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_10      0x001
//! Audio in/out unbalanced (RCA connector) high impedance at -4db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_UNBALANCED_4       0x002
//! Audio in/out balanced (XLR connector) 600ohm impedance at -10db
(cmdGetSetValue::gsAudInSelect cmdGetSetValue::gsAudOutSelect)
#define GS_AUDSELECT_BALANCED_10        0x010
//! Audio in/out balanced (XLR connector) 600ohm impedance 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
//! 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

//! 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

#define GS_MPEG_RESOLUTION_120      0x001
#define GS_MPEG_RESOLUTION_240      0x002
#define GS_MPEG_RESOLUTION_288      0x004
#define GS_MPEG_RESOLUTION_352      0x008
#define GS_MPEG_RESOLUTION_480      0x010
#define GS_MPEG_RESOLUTION_512      0x020
#define GS_MPEG_RESOLUTION_544      0x040
#define GS_MPEG_RESOLUTION_576      0x080
#define GS_MPEG_RESOLUTION_608      0x100
#define GS_MPEG_RESOLUTION_704      0x200
#define GS_MPEG_RESOLUTION_720      0x400

#define GS_MPEG_CHROMA_FORMAT_420    0x1
#define GS_MPEG_CHROMA_FORMAT_422    0x2
#define GS_MPEG_CHROMA_FORMAT_444    0x4

#define GS_MPEG_DC_PRECISION_8       0x1
#define GS_MPEG_DC_PRECISION_9       0x2
#define GS_MPEG_DC_PRECISION_10      0x4
#define GS_MPEG_DC_PRECISION_11      0x8

#define GS_MPEG_ASPECT_RATIO_SQUARE  0x1
#define GS_MPEG_ASPECT_RATIO_4x3     0x2
#define GS_MPEG_ASPECT_RATIO_16x9    0x4
#define GS_MPEG_ASPECT_RATIO_2_21x1  0x8

#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_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

//! Signal format NTSC square pixel (320x240 or 640x480) @ 29.97 or 30 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_NTSC                    0x0001
//!#define GS_SIGFORM_525                    0x0001
//! Signal format PAL square pixel (320x288 or 640x576) @ 25 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_PAL                      0x0002
//! Signal format NTSC square pixel (360/352x243/240 or 720/704x486/480) @ 29.97 or
30 fps gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CCIR_NTSC                0x0004
//! Signal format PAL square pixel (360/352x288 or 720/704x576) @ 25 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CCIR_PAL                  0x0008
//! 2200x1125 raster, 1920x1035 production aperture (1888x1017 clean) @ 30 fps
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1035i_30_260M            0x000100
//! 2200x1125 raster, 1920x1035 production aperture (1888x1017 clean) @ 29.97 fp
gsGetSetValue::gsSignalFormats
#define GS_SIGFORM_1035i_30X_260M           0x000200
//! 1920x1080i (274M-1997 Table1 System 4) @ 29.97 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_30                  0x000400 /* (274M-
1997 Table1 System 4) */
//! 1920x1080i (274M-1997 Table1 System 4) @ 30 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_30X                0x000800 /* (274M-1997
Table1 System 5) */
//! 1920x1080i (274M-1997 Table1 System 4) @ 25 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_25                  0x001000 /* (274M-
1997 Table1 System 6) */
//! 1920x1080sf (274M-1997 Table1 System 4) @ 24 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_24                  0x002000

```

```

//! 1920x1080sf (274M-1997 Table1 System 4) @ 23.98 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080i_24X          0x004000
//! 1920x1080P (274M-1997 Table1 System 4) @ 30 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_30          0x008000 /* (274M-1997
Table1 System 7) */
//! 1920x1080P (274M-1997 Table1 System 4) @ 29.97 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_30X        0x010000 /* (274M-
1997 Table1 System 8) */
//! 1920x1080P (274M-1997 Table1 System 4) @ 25 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_25          0x020000 /* (274M-1997
Table1 System 9) */
//! 1920x1080P (274M-1997 Table1 System 4) @ 24 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_24          0x040000 /* (274M-1997
Table1 System 10) */
//! 1920x1080P (274M-1997 Table1 System 4) @ 23.98 gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_1080_24X        0x080000 /* (274M-
1997 Table1 System 11) */
//! 1650x750 raster, 1280x720 production aperture (1248x702 clean): @ 60
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_720_60          0x100000 /* (296M-1996
Table1 System 1) */
//! 1650x750 raster, 1280x720 production aperture (1248x702 clean): @ 59.94
gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_720_60X        0x200000 /* (296M-1996
Table1 System 2) */
//! All non video rate types (eg. 15fps, 10fps, 37fps) gsGetSetValue::gsSignalFormat
#define GS_SIGFORM_CUSTOM          0x10000000

//! 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
//! Wavelet hardware codec (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_WAVELET        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_HDSOY        0x00020000
//! Uncompressed Y'CrCb High Def Video (gsGetSetValue::gsCompType)
#define GS_COMPTYPE_HDUNCOMP     0x00100000

/** @{ */
/** 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 seperate 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 necessary (gsGetSetValue::gsPBEE)
#define GS_PBEE_AUTO                0x00000000
//! Allow playback only output - no passthrough (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
//! Use device default head (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

//! Command is not supported see cmdType::ctGetValue, cmdType::ctSetValue,
cmdType::ctValueSupported
#define GS_NOT_SUPPORTED 0xFFFFFFFF
/**
 * Parameter is bad see cmdType::ctGetValue, cmdType::ctSetValue,
cmdType::ctValueSupported and
 * MEDIACMD::dwPosition, MEDIACMD::dwStart and MEDIACMD::dwEnd
 */
#define GS_BAD_PARAM            0xFFFFFFF0
/**
 * False for boolean cmdType::ctGetValue, cmdType::ctSetValue
 */
#define GS_FALSE                0x00
/**
 * True for boolean cmdType::ctGetValue, cmdType::ctSetValue
 */
#define GS_TRUE                  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

// 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
    //! INTERNAL dlist.dll Link List Pointers - do not use
    void * pPrev; // Next Inode
    //! INTERNAL dlist.dll Link List Pointers - do not use
    void * pNext; // Prev Inode
    //! INTERNAL dlist.dll Link List Pointers - do not use
    void * pList; // Parent or List owner

    /**
     * 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 #ctFlags 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::Vtc, 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 immediately 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

```



```

        __mCmd_.dwEnd = TC_ILLEGAL;    }
    //! Initailize a media cmd pointer to all illegal (no command)
    #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_)->ISpeed = SPD_ILLEGAL;
        \
        ((PMEDIACMD)__mCmd_)->dwVideoChannels = vidChanAll;    \
        ((PMEDIACMD)__mCmd_)->dwAudioChannels = audChanAll;    \
        ((PMEDIACMD)__mCmd_)->dwInfoChannels = infChanAll;    \
        ((PMEDIACMD)__mCmd_)->dwCmdAlt = CHAN_ILLEGAL;
        \
        ((PMEDIACMD)__mCmd_)->dwPosition = TC_ILLEGAL;    \
        ((PMEDIACMD)__mCmd_)->dwStart = TC_ILLEGAL;
        \
        ((PMEDIACMD)__mCmd_)->dwEnd = TC_ILLEGAL;    }

```

```
#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.

<BR>

<BR>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.

<BR>

<BR>Physical Transport

<BR>

<BR>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>

```

<BR> The structure:  MEDIACMD
<BR> The commands:  #cmdType
<BR> The Flags:     #cmdFlags
<BR> Channels:      #cmdVidChan, #cmdAudChan, #cmdinf
<BR> GetSet Commands #cmdGetSetValue
<BR>   Basic       cmdGetSetValue::gsTc
<BR>   Clip        cmdGetSetValue::gsGetNextClip
<BR>   Channel     cmdGetSetValue::gsAudChan
<BR>   Audio       cmdGetSetValue::gsAudInSelect
<BR>   Video General cmdGetSetValue::gsVidFreeze
<BR>   Video Input  cmdGetSetValue::gsVidInSelect
<BR>   Video TBC   cmdGetSetValue::gsVidSetup
<BR>   Video Output cmdGetSetValue::gsVidOutSelect
<BR>   Signal Type  cmdGetSetValue::gsSignalFormat
<BR>   Compression  cmdGetSetValue::gsCompType
<BR>   Storage     cmdGetSetValue::gsTotalStorageAvail
<BR>   Control     cmdGetSetValue::gsLocal
<BR>   Info        cmdGetSetValue::gsVVWVersion
<BR>   Internal    cmdGetSetValue::gsSetHwnds
<BR>   Mode+Info   cmdGetSetValue::gsChannelExist

```

```

<BR> Returns #GS_NOT_SUPPORTED
<BR> Declaration Local #DECLARE_MEDIACMD
<BR> Init Local #INIT_MEDIACMD
<BR> Init Pointer #INIT_PMEDIACMD
<BR> Sizes #SIZEOF_MEDIACMD_BASE
<BR>
</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 <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
<BR>
\b Simple \b Pause <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctPause<BR>
<BR>
\b Simple \b Stop <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctStop<BR>
<BR>
\b Play \b From-To <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd<BR>
MEDIACMD::dwStart = 1800 // 1 minute in frames NTSC NDF<BR>
MEDIACMD::dwEnd = 2100 // 1 minute 10 seconds NTSC NDF<BR>
<BR>
\b Play \b DMC \b (Play \b at \b speed) <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 36<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
MEDIACMD::cfFlags = #cfUseSpeed<BR>
MEDIACMD::lSpeed = 32760 // Half play speed forward<BR>
<BR>
\b Record \b Edit <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctRecord<BR>
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfUsePresets<BR>

```

```

MEDIACMD::dwVideoChannels = 0x01           // Record V<BR>
MEDIACMD::dwAudioChannels = 0x03         // Record A1 & A2 (AA)<BR>
MEDIACMD::dwInfoChannels = 0x00<BR>
MEDIACMD::dwStart = 3000<BR>
MEDIACMD::dwEnd = 3200<BR>
<BR>
\b Play \b Two \b Segments <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd<BR>
MEDIACMD::dwStart = 2200<BR>
MEDIACMD::dwEnd = 2500<BR>
<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfDeferred<BR>
MEDIACMD::dwStart = 5300<BR>
MEDIACMD::dwEnd = 5450<BR>
<BR>
\b Play \b A \b Named \b Clip <BR>
Char szName = "C:\Adir\Afile.OMF"<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 + strlen(szName)<BR>
MEDIACMD::ctCmd = #ctPlay<BR>
MEDIACMD::cfFlags = #cfUseClipID<BR>
strcpy( MEDIACMD::arbID , szName)<BR>
<BR>
\b Seek \b To \b 1 \b Minute <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 48<BR>
MEDIACMD::ctCmd = #ctPause<BR>
MEDIACMD::cfFlags = #cfUsePosition<BR>
MEDIACMD::dwPosition = 1800           // Frames NTSC NDF<BR>
<BR>
\b Step \b Reverse \b 1 \b Frame <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 48<BR>
MEDIACMD::ctCmd = #ctPause<BR>
MEDIACMD::cfFlags = #cfUsePositionOffset<BR>
MEDIACMD::dwPosition = (DWORD) -1<BR>
<BR>
\b Record \b A \b 1 \b Minute \b Named \b Clip <BR>
Char szName = "C:\Adir\ArecordFile.MOV"<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 + strlen(szName)<BR>
MEDIACMD::ctCmd = #ctRecord<BR>
MEDIACMD::cfFlags = #cfUseClipID | #cfUseEnd<BR>

```

```

MEDIACMD::dwEnd = 1800<BR>
strcpy( MEDIACMD::arbID , szName)<BR>
<BR>
\b Record \b An \b Odetics \b or \b Louth \b Clip \b (ext) <BR>
(Note: The stop command is required for accuracy on some DDRs)<BR>
Char szName = "THISCLIP" // Max 8 char<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 + strlen(szName)<BR>
MEDIACMD::ctCmd = #ctStop // Record ready<BR>
MEDIACMD::cfFlags = #cfUseClipID<BR>
strcpy( MEDIACMD::arbID , szName)<BR>
<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 + strlen(szName)<BR>
MEDIACMD::ctCmd = #ctRecord<BR>
MEDIACMD::cfFlags = #cfUseClipID<BR>
strcpy( MEDIACMD::arbID , szName)<BR>
<BR>
\b Eject \b The \b Current \b Media <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctEject<BR>
<BR>
\b Setup \b The \b Video \b To \b Nominal <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctSetValue<BR>
MEDIACMD::dwCmdAlt = gsVidSetup<BR>
MEDIACMD::dwStart = 0<BR>
<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctSetValue<BR>
MEDIACMD::dwCmdAlt = gsVidVideo<BR>
MEDIACMD::dwStart = 0<BR>
<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctSetValue<BR>
MEDIACMD::dwCmdAlt = gsVidHue<BR>
MEDIACMD::dwStart = 0<BR>
<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctSetValue<BR>
MEDIACMD::dwCmdAlt = gsVidChroma<BR>
MEDIACMD::dwStart = 0<BR>
<BR>
\b Check \b The \b Device \b Type <BR>

```

```

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctGetValue<BR>
MEDIACMD::dwCmdAlt = gsVtrType<BR>
MEDIACMD::dwStart = 0<BR>
(Returns: Vtr/Ddr/Media type in .dwStart)<BR>
<BR>
\b Change \b The \b Default \b TC \b Type \b To \b VITC <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctSetValue<BR>
MEDIACMD::dwCmdAlt = gsVitcTc<BR>
MEDIACMD::dwStart = GS_DEFAULT<BR>
<BR>
\b Transfer \b From \b External \b VTR \b To \b Internal \b Channel <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctTransfer<BR>
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd | #cfUsePresets<BR>
MEDIACMD::dwVideoChannels = 0x01 // Capture Video<BR>
MEDIACMD::dwAudioChannels = 0x00 // No Audio<BR>
MEDIACMD::dwInfoChannels = 0x00<BR>
MEDIACMD::dwCmdAlt = hExternalChannel // The VTR channel<BR>
MEDIACMD::dwPosition = 2100 // Where to record to<BR>
MEDIACMD::dwStart = 10850 // Source In on VTR<BR>
MEDIACMD::dwEnd = 11000 // Source Out on
VTR<BR>
<BR>
\b Insert \b Media \b From \b File \b Over \b Current <BR>
Char szName = "C:\Adir\ArecordFile.MOV"<BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 + strlen(szName)<BR>
MEDIACMD::ctCmd = #ctInsert<BR>
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd | #cfUseClipID<BR>
MEDIACMD::dwPosition = 2100 // Insert @ on target channel<BR>
MEDIACMD::dwStart = 0 // Start on source file<BR>
MEDIACMD::dwEnd = 180 // End on source file<BR>
strcpy( MEDIACMD::arbID , szName) // Name of source file<BR>
<BR>
\b Delete \b A \b Section \b Of \b Media \b (No \b Hole) <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60<BR>
MEDIACMD::ctCmd = #ctErase<BR>
MEDIACMD::cfFlags = #cfUseStart | #cfUseEnd | #cfRipple // Remove #cfRipple to
leave hole<BR>
MEDIACMD::dwStart = 140500 // Start on target media<BR>
MEDIACMD::dwEnd = 150010 // End on target media<BR>
<BR>
\b Trim \b A \b Clip <BR>

```

```

MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 60 <BR>
MEDIACMD::ctCmd = #ctTrim<BR>
MEDIACMD::cfFlags = #cfUsePosition | #cfUseStart | #cfUseEnd <BR>
MEDIACMD::dwPosition = 2100 // Clip @ position<BR>
MEDIACMD::dwStart = +32 // Clip 32 frames from
start<BR>
MEDIACMD::dwEnd = (DWORD) -12 // Clip 12 frame from end<BR>
<BR>
\b Terminate \b Session \b (Restart \b At \b Default) <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctTerminate<BR>
<BR>
\b Abort \b Current \b Command <BR>
MEDIACMD::dwCmdID = #MEDIACMD_CURRENT<BR>
MEDIACMD::dwStructSize = 28<BR>
MEDIACMD::ctCmd = #ctAbort<BR>
<BR>

* \section mediacmdsamplertns MEDIACMD Sample Returns
Sample Returns

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

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

*/
#endif //_MEDIACMD_INCLUDED_H

```

## Appendix II – MediaCmd.bas

```
Attribute VB_Name = "VvwMediaCmd"
Option Explicit

'=====
'
' The channel group definitions
'
'=====

Public Const nVvWSync As Long = 65536
Public Const nVvWMaxChannels As Integer = 255
Public Const nVvWMaxInternal As Integer = 63
Public Const nVvWMaxExternal As Integer = 127
Public Const nVvWMaxNetwork As Integer = 255
'Public Const nVvWMaxChannels As Integer = nVvWMaxNetwork

'=====
'
' The main command type definition
'
'=====

Public Const CMD_MAX_CLIP_ID_LEN As Long = 268 ' _MAX_PATH + 8
Public Const CMD_MAX_CLIP_ID_INTERN_ALLOC As Long =
CMD_MAX_CLIP_ID_LEN + 4
'
' The command filled in to send to the VVW pipe
'
Type MEDIACMD
    dwResP As Long ' Reserved
    dwResL As Long ' Reserved
    dwResN As Long ' Reserved
    dwCmdID As Long ' = MEDIACMD_CURRENT
(version info)
    dwStructSize As Long ' MEDIACMD_SIZE + len
(szClipName)
    dwChannel As Long ' VVW Channel
    ctCmd As Long ' A ct... (eg. ctPlay)
    cfFlags As Long ' A cf... bit array
(ctUsePosition Or cfDefered)
    lSpeed As Long ' A speed (65520 = play)
    dwVideoChannels As Long ' Bit array, video
channels to use
    dwAudioChannels As Long ' Bit array, audio
channels to use
```

```

        dwInfoChannels As Long          ' Bit array, info
channels to use
        dwCmdAlt As Long                ' Time to do command
(frames, ms, offset...) or source channel
        dwPosition As Long              ' Current position or
seek to
        dwStart As Long                 ' Start of event
        dwEnd As Long                   ' End of event
        szClipName As String * CMD_MAX_CLIP_ID_INTERN_ALLOC
,

Multipurpose string
End Type

'=====
'
' Constants to use for specific type elements
'
'=====
'
' Defines for the (MEDIACMD.dwCmdID)
'
Public Const MEDIACMD_VERSION_MAJOR As Long = &H101
Public Const MEDIACMD_VERSION_MINOR As Long = &H3
Public Const MEDIACMD_VERSION_MASK As Long = &HFFFF
Public Const MEDIACMD_CHECK_VER As Long = &HFA250000
Public Const MEDIACMD_CHECK_MASK As Long = &HFFFF0000

Public Const MEDIACMD_CURRENT As Long = MEDIACMD_VERSION_MAJOR Or
MEDIACMD_VERSION_MINOR Or MEDIACMD_CHECK_VER

'
' Basic size (without szClipID) or the MEDIACMD type
'
Public Const SIZEOF_MEDIACMD As Long = 64

'
' Accepted commands for the VVW (MEDIACMD.ctCmd)
'
Public Const ctStop As Long = 0
Public Const ctPause As Long = 1 + ctStop
Public Const ctPlay As Long = 1 + ctPause
Public Const ctRecord As Long = 1 + ctPlay
Public Const ctRecStop As Long = 1 + ctRecord
Public Const ctEject As Long = 1 + ctRecStop
Public Const ctTransfer As Long = 1 + ctEject
Public Const ctInsert As Long = 1 + ctTransfer
Public Const ctBlank As Long = 1 + ctInsert
Public Const ctDelete As Long = 1 + ctBlank

```

```

Public Const ctTrim As Long = 1 + ctDelete
Public Const ctChanSelect As Long = 1 + ctTrim
Public Const ctGetState As Long = 1 + ctChanSelect
Public Const ctSetState As Long = 1 + ctGetState
Public Const ctGetValue As Long = 1 + ctSetState
Public Const ctSetValue As Long = 1 + ctGetValue
Public Const ctValueSupported As Long = 1 + ctSetValue
Public Const ctError As Long = 1 + ctValueSupported
Public Const ctTerminate As Long = 1 + ctError
Public Const ctAbort As Long = 1 + ctTerminate

'
' Accepted flags (MEDIACMD.cfFlags)
' These should be 'Or'ed together as a bit field
'
Public Const cfDeferred As Long = &H1
Public Const cfTimeMs As Long = &H2
Public Const cfTimeTarget As Long = &H4
Public Const cfTimeHouseClock As Long = &H8
Public Const cfUseSpeed As Long = &H10
Public Const cfUsePresets As Long = &H20
Public Const cfUsePosition As Long = &H40
Public Const cfUsePositionOffset As Long = &H80
Public Const cfUseStart As Long = &H100
Public Const cfUseStartOffset As Long = &H200
Public Const cfUseEnd As Long = &H400
Public Const cfUseEndOffset As Long = &H800
Public Const cfUseAllIDs As Long = &H1000
Public Const cfUseClipID As Long = &H2000
Public Const cfNoClipFiles As Long = &H4000
Public Const cfNoTCSpaces As Long = &H8000
Public Const cfUseCmdAlt As Long = &H10000
Public Const cfIsShuttle As Long = &H20000
' Starts bit 20
Public Const cfFields As Long = &H100000
Public Const cfRipple As Long = &H200000
Public Const cfLoop As Long = &H400000
Public Const cfTrigger As Long = &H800000
' Starts bit 28
Public Const cfInvert As Long = &H10000000
Public Const cfTest As Long = &H20000000
Public Const cfOverrideDeferred As Long = &H40000000
Public Const cfNoReturn As Long = &H80000000

'
' (MEDIACMD.lSpeed)
' Speeds may be any value, but here are some handy defaults
'
Public Const SpeedFwdPlay As Long = 65520

```

```

Public Const SpeedPlay As Long = SpeedFwdPlay
Public Const SpeedPause As Long = 0
Public Const SpeedRevPlay As Long = -65520
Public Const SpeedFwdMax As Long = 5896800
Public Const SpeedRevMax As Long = -SpeedFwdMax
' Illegal speed notation
Public Const SpeedILLEGAL As Long = 2147483647

'
' (MEDIACMD.dwVideoChannels)
' Video channels are a bit field where each bit represents
' a channel starting at 0 and going to 31
'
Public Const vidChanAll As Long = &FFFFFFFF
Public Const vidChanNone As Long = 0
Public Const vidChan1 As Long = &H1

'
' (MEDIACMD.dwAudioChannels)
' Audio channels are a bit field where each bit represents
' a channel starting at 0 and going to 31
'
Public Const audChanAll As Long = vidChanAll
Public Const audChanNone As Long = vidChanNone
Public Const audChan1 As Long = vidChan1
Public Const audChan2 As Long = &H2
Public Const audChan3 As Long = &H4
Public Const audChan4 As Long = &H8
Public Const audChanStereo1 As Long = audChan1 Or audChan2

'
' (MEDIACMD.dwInfoChannels)
'
Public Const infChanAll As Long = vidChanAll
Public Const infLtc As Long = &H1
Public Const infVitc As Long = &H2
Public Const infSrcCtl As Long = &H4
Public Const infSrcLtc As Long = &H8
Public Const infSrcVitc As Long = &H10
Public Const infRecTime As Long = &H20
Public Const infRecDate As Long = &H40
Public Const infCC As Long = &H80
Public Const infAuth As Long = &H100
Public Const infCopyright As Long = &H200
Public Const infOwner As Long = &H1
Public Const infSourceName As Long = &H1
Public Const infProxyNames As Long = &H1
'

```

```

' (MEDIACMD.dwCmdAlt)
' (MEDIACMD.dwPosition)
' (MEDIACMD.dwStart)
' (MEDIACMD.dwEnd)
' Each of the paramters above indicate a time by frames
' (or fields/milliseconds/offset depending on flags)
' Other then a valid time, the following may be used
'
Public Const TCILLEGAL As Long = &HFFFFFFF

' (MEDID_CMD.szClipName)
' Used by some commands for clip handling
'

'=====
'
' Get/Set values supported for the commands
'      GetValue, SetValue, ValueSupported
'
'=====
'Timecode Stuff
Public Const gsTc As Long = 1
Public Const gsUb As Long = gsTc + 1           'Current user
bits
Public Const gsLtcTc As Long = gsUb + 1       'Current LTC
TC
Public Const gsLtcUb As Long = gsLtcTc + 1    'Current LTC
user bits
Public Const gsVitcTc As Long = gsLtcUb + 1   'Current VITC
TC
Public Const gsVitcUb As Long = gsVitcTc + 1  'Current VITC
user bits
Public Const gsTcSource As Long = gsVitcUb + 1 'Default TC
Source
Public Const GS_TCSOURCE_LTC As Long = 1
Public Const GS_TCSOURCE_VITC As Long = 2
Public Const GS_TCSOURCE_CTL As Long = 4
Public Const GS_TCSOURCE_CLIP As Long = 7
'---
Public Const gsTcType As Long = gsTcSource + 1 'Default time
code type
Public Const GS_TCTYPE_FILM As Long = &H1     ' 24 fps
Public Const GS_TCTYPE_NDF As Long = &H2     ' 25 fps
Public Const GS_TCTYPE_DF As Long = &H4      ' 29.97
Dropping Frames
Public Const GS_TCTYPE_PAL As Long = &H8     ' 29.97 No
Dropping
Public Const gsStart As Long = gsTcType + 1   'Lowest
possible TC

```

```

Public Const gsEnd As Long = gsStart + 1           'Highest
possible TC
Public Const gsIn As Long = gsEnd + 1             'Current in
point
Public Const gsLastIn As Long = gsIn + 1         'Last in point
Public Const gsOut As Long = gsLastIn + 1        'Current out
point
Public Const gsLastOut As Long = gsOut + 1       'Last out
point
Public Const gsEditOn As Long = gsLastOut + 1    '
Public Const gsEditOff As Long = gsEditOn + 1    '
Public Const gsPreroll As Long = gsEditOff + 1   '
Public Const gsPostroll As Long = gsPreroll + 1  '
Public Const gsAutoMode As Long = gsPostroll + 1 '
Public Const gsPlayDelay As Long = gsAutoMode + 1 '

' Clip stuff
Public Const gsGetNextClip As Long = 90          ' see H file
Public Const gsFirstClip As Long = gsGetNextClip + 1 ' First
clip name, status, dwStart, dwEnd
Public Const gsNextClip As Long = gsFirstClip + 1 'Next clip
name, status, dwStart, dwEnd
Public Const gsTCSGetTLClipState As Long = gsNextClip + 1
Public Const gsTCSGetTLClipInfo As Long = gsTCSGetTLClipState + 1
Public Const gsClipInfo As Long = gsTCSGetTLClipInfo + 1 'Same
as above for named clip
Public Const gsClipCopy As Long = gsClipInfo + 1 'Copy current
clip to new name

'Edit Preset Stuff
Public Const gsAudChan As Long = 100
Public Const gsVidChan As Long = gsAudChan + 1   'Available
video channels
Public Const gsInfChan As Long = gsVidChan + 1
Public Const gsAudSelect As Long = gsInfChan + 1  'Selected
audio channels
Public Const gsVidSelect As Long = gsAudSelect + 1 'Selected
video channels
Public Const gsInfSelect As Long = gsVidSelect + 1
Public Const gsAudEdit As Long = gsInfSelect + 1  'Edit ready
audio channels
Public Const gsVidEdit As Long = gsAudEdit + 1    'Edit ready
video channels
Public Const gsInfEdit As Long = gsVidEdit + 1

Public Const gsAudInSelect As Long = 200         '
Public Const gsAudOutSelect As Long = gsAudInSelect + 1
Public Const GS_AUDSELECT_UNBALANCED_10 As Long = &H1
Public Const GS_AUDSELECT_UNBALANCED_4 As Long = &H2

```

```
Public Const GS_AUDSELECT_BALANCED_10 As Long = &H10
Public Const GS_AUDSELECT_BALANCED_4 As Long = &H20
Public Const GS_AUDSELECT_SPDIF As Long = &H100
Public Const GS_AUDSELECT_AES_EBU As Long = &H200
Public Const GS_AUDSELECT_EMBEDDED As Long = &H400
Public Const GS_AUDSELECT_AES_EBU_PRO As Long = &H800
Public Const GS_AUDSELECT_NONE As Long = &H0
Public Const GS_AUDSELECT_SILENT As Long = &H40
Public Const gsAudInputLevel As Long = gsAudOutSelect + 1
'Input level setting
Public Const gsAudOutputLevel As Long = gsAudInputLevel + 1
'Ouput level setting
Public Const gsAudAdvanceLevel As Long = gsAudOutputLevel + 1
Public Const gsAudOutPhase As Long = gsAudAdvanceLevel + 1
Public Const gsAudOutAdvancePhase As Long = gsAudOutPhase + 1
Public Const gsAudCrossFadeTime As Long = gsAudOutAdvancePhase + 1
Public Const gsAudLtcEnable As Long = gsAudCrossFadeTime + 1
Public Const gsAudInLtcChannel As Long = gsAudLtcEnable + 1
Public Const gsAudOutLtcChannel As Long = gsAudInLtcChannel + 1
Public Const gsAudDtmfEnable As Long = gsAudOutLtcChannel + 1
Public Const gsAudInDtmfChannel As Long = gsAudDtmfEnable + 1
Public Const gsAudOutDtmfChannel As Long = gsAudInDtmfChannel + 1
Public Const gsAudWavePeakRMS As Long = gsAudOutDtmfChannel + 1
Public Const gsAudInputBitRate As Long = gsAudWavePeakRMS + 1
'Defines for various audio bit rates.
Public Const GS_AUD_BIT_RATE_32000 As Long = &H1
Public Const GS_AUD_BIT_RATE_41100 As Long = &H2
Public Const GS_AUD_BIT_RATE_48000 As Long = &H4
Public Const GS_AUD_BIT_RATE_56000 As Long = &H8
Public Const GS_AUD_BIT_RATE_64000 As Long = &H10
Public Const GS_AUD_BIT_RATE_80000 As Long = &H20
Public Const GS_AUD_BIT_RATE_96000 As Long = &H40
Public Const GS_AUD_BIT_RATE_112000 As Long = &H80
Public Const GS_AUD_BIT_RATE_128000 As Long = &H100
Public Const GS_AUD_BIT_RATE_160000 As Long = &H200
Public Const GS_AUD_BIT_RATE_192000 As Long = &H400
Public Const GS_AUD_BIT_RATE_224000 As Long = &H800
Public Const GS_AUD_BIT_RATE_256000 As Long = &H1000
Public Const GS_AUD_BIT_RATE_320000 As Long = &H2000
Public Const GS_AUD_BIT_RATE_384000 As Long = &H4000
Public Const gsAudInputSampleRate As Long = gsAudInputBitRate + 1
'use above defined
Public Const gsAudInputMode As Long = gsAudInputSampleRate + 1
Public Const GS_AUD_STEREO As Long = &H1
Public Const GS_AUD_JOINT_STEREO As Long = &H2
Public Const GS_AUD_DUAL As Long = &H4
Public Const GS_AUD_SINGLE As Long = &H8
Public Const GS_AUD_MULTIPLE As Long = &H10
Public Const gsAudInputHeadRoom As Long = gsAudInputMode + 1
```

```

Public Const GS_AUD_HEADROOM_18 As Long = &H1
Public Const GS_AUD_HEADROOM_20 As Long = &H2
Public Const gsAudInputOriginal As Long = gsAudInputHeadRoom + 1
Public Const gsAudInputErrorProtect As Long = gsAudInputOriginal
+ 1
Public Const gsAudInputCopyright As Long = gsAudInputErrorProtect
+ 1
Public Const gsAudInputSlave As Long = gsAudInputCopyright + 1

' General Video
Public Const gsVidFreeze As Long = 300
Public Const gsVidPreReadMode As Long = gsVidFreeze + 1
Public Const gsVidEditField As Long = gsVidPreReadMode + 1
Public Const gsVidRecFrame As Long = gsVidEditField + 1
Public Const gsVidPlayFrame As Long = gsVidRecFrame + 1
Public Const gsVidNoEE As Long = gsVidPlayFrame + 1
Public Const gsVidSuperimpose As Long = gsVidNoEE + 1

' Video Input
Public Const gsVidInSelect As Long = 400
Public Const GS_VIDSELECT_COMPOSITE As Long = &H1
Public Const GS_VIDSELECT_SVIDEO As Long = &H2
Public Const GS_VIDSELECT_COMPONENT_YUV As Long = &H10
Public Const GS_VIDSELECT_COMPONENT_YUV_M2 As Long = &H20
Public Const GS_VIDSELECT_COMPONENT_YUV_SMPTE As Long = &H40
Public Const GS_VIDSELECT_COMPONENT_RGB As Long = &H80
Public Const GS_VIDSELECT_D1_SERIAL As Long = &H100
Public Const GS_VIDSELECT_D1_PARALLEL As Long = &H200
Public Const GS_VIDSELECT_NONE As Long = &H0
'---
Public Const gsVidInLockType As Long = gsVidInSelect + 1
Public Const GS_VIDLOCKTYPE_VTR As Long = &H1
Public Const GS_VIDLOCKTYPE_BROADCAST As Long = &H2
'---
Public Const gsVidInSetup As Long = gsVidInLockType + 1
Public Const gsVidInVideo As Long = gsVidInSetup + 1
Public Const gsVidInHue As Long = gsVidInVideo + 1
Public Const gsVidInChroma As Long = gsVidInHue + 1
Public Const gsVidInUChroma As Long = gsVidInChroma + 1
Public Const gsVidInVChroma As Long = gsVidInUChroma + 1
Public Const gsVidInColorKiller As Long = gsVidInVChroma + 1
Public Const gsVidInAGC As Long = gsVidInColorKiller + 1
Public Const gsVidInBandwidth As Long = gsVidInAGC + 1
Public Const GS_VIDBAND_STANDARD As Long = &H1
Public Const GS_VIDBAND_MEDIUM As Long = &H2
Public Const GS_VIDBAND_HIGH As Long = &H4
Public Const GS_VIDBAND_NOTCH As Long = &H8
'---
Public Const gsVidInBlack As Long = gsVidInBandwidth + 1

```

```

Public Const GS_VIDBLACK_SETUP As Long = &H1
Public Const GS_VIDBLACK_CRYSTAL As Long = &H2
Public Const GS_VIDBLACK_SUPER As Long = &H4
'---

Public Const gsVidInWhite As Long = gsVidInBlack + 1
Public Const GS_VIDWHITE_CLAMP As Long = &H1
Public Const GS_VIDWHITE_SCALE As Long = &H2
Public Const GS_VIDWHITE_FREE As Long = &H4
'---

Public Const gsVidInCoring As Long = gsVidInWhite + 1
Public Const gsVidInPeaking As Long = gsVidInCoring + 1
Public Const gsVidInSharpness As Long = gsVidInPeaking + 1
Public Const gsVidInGamma As Long = gsVidInSharpness + 1

' Video TBC (internal)
Public Const gsVidSetup As Long = 500
Public Const gsVidVideo As Long = gsVidSetup + 1
Public Const gsVidHue As Long = gsVidVideo + 1
Public Const gsVidChroma As Long = gsVidHue + 1
Public Const gsVidUChroma As Long = gsVidChroma + 1
Public Const gsVidVChroma As Long = gsVidUChroma + 1
Public Const gsVidBandwidth As Long = gsVidVChroma + 1
' See gsVidInBandwidth
Public Const gsVidBlackSetup As Long = gsVidBandwidth + 1
Public Const gsVidColor As Long = gsVidBlackSetup + 1

' Video Output
Public Const gsVidOutSelect As Long = 600
Public Const gsVidOutGenlock As Long = gsVidOutSelect + 1
Public Const gsVidOutGenlockSource As Long = gsVidOutGenlock + 1
Public Const GS_LOCKSRC_NONE As Long = &H1
Public Const GS_LOCKSRC_EXTIN As Long = &H2
Public Const GS_LOCKSRC_INPUT As Long = &H4
Public Const GS_LOCKSRC_CVBS As Long = &H8
Public Const GS_LOCKSRC_SVIDEO As Long = &H10
Public Const GS_LOCKSRC_IN_Y As Long = &H20
Public Const GS_LOCKSRC_SDI As Long = &H40
'---

Public Const gsVidOutLockType As Long = gsVidOutGenlockSource + 1
Public Const gsVidOutHPhase As Long = gsVidOutLockType + 1
Public Const gsVidOutSubCarrier As Long = gsVidOutHPhase + 1
Public Const gsVidOutCoring As Long = gsVidOutSubCarrier + 1
Public Const gsVidOutPeaking As Long = gsVidOutCoring + 1
Public Const gsVidOutAdjust1 As Long = gsVidOutPeaking + 1
Public Const gsVidOutAdjust2 As Long = gsVidOutAdjust1 + 1
Public Const gsVidOutGenlockDelay As Long = gsVidOutAdjust2 + 1
'MPEG Specific
Public Const gsMpegVerticalRes As Long = 700
Public Const gsMpegHorizontalRes As Long = gsMpegVerticalRes + 1

```

```
Public Const GS_MPEG_RESOLUTION_120 As Long = &H1
Public Const GS_MPEG_RESOLUTION_240 As Long = &H2
Public Const GS_MPEG_RESOLUTION_288 As Long = &H4
Public Const GS_MPEG_RESOLUTION_352 As Long = &H8
Public Const GS_MPEG_RESOLUTION_480 As Long = &H10
Public Const GS_MPEG_RESOLUTION_512 As Long = &H20
Public Const GS_MPEG_RESOLUTION_544 As Long = &H40
Public Const GS_MPEG_RESOLUTION_576 As Long = &H80
Public Const GS_MPEG_RESOLUTION_608 As Long = &H100
Public Const GS_MPEG_RESOLUTION_704 As Long = &H200
Public Const GS_MPEG_RESOLUTION_720 As Long = &H400
Public Const gsMpegChromaFormat As Long = gsMpegHorizontalRes + 1
Public Const GS_MPEG_CHROMA_FORMAT_420 As Long = &H1
Public Const GS_MPEG_CHROMA_FORMAT_422 As Long = &H2
Public Const GS_MPEG_CHROMA_FORMAT_444 As Long = &H4
Public Const gsMpegDCPrecision As Long = gsMpegChromaFormat + 1
Public Const GS_MPEG_DC_PRECISION_8 As Long = &H1
Public Const GS_MPEG_DC_PRECISION_9 As Long = &H2
Public Const GS_MPEG_DC_PRECISION_10 As Long = &H4
Public Const GS_MPEG_DC_PRECISION_11 As Long = &H8
Public Const gsMpegAspectRatio As Long = gsMpegDCPrecision + 1
Public Const GS_MPEG_ASPECT_RATIO_SQUARE As Long = &H1
Public Const GS_MPEG_ASPECT_RATIO_4x3 As Long = &H2
Public Const GS_MPEG_ASPECT_RATIO_16x9 As Long = &H4
Public Const GS_MPEG_ASPECT_RATIO_2_21x1 As Long = &H8
Public Const gsMpegStandard As Long = gsMpegAspectRatio + 1
Public Const GS_MPEG_STANDARD_SYSTEM As Long = &H1
Public Const GS_MPEG_STANDARD_PROGRAM As Long = &H2
Public Const GS_MPEG_STANDARD_TRANSPORT As Long = &H4
Public Const GS_MPEG_STANDARD_ELEMENTARY As Long = &H8
Public Const gsMpegLanguageCode As Long = gsMpegStandard + 1
Public Const GS_MPEG_LANGAUGE_ENGLISH As Long = &H1
Public Const GS_MPEG_LANGAUGE_SPANISH As Long = &H2
Public Const GS_MPEG_LANGAUGE_FRENCH As Long = &H4
Public Const GS_MPEG_LANGAUGE_GERMAN As Long = &H8
Public Const GS_MPEG_LANGAUGE_JAPANESE As Long = &H10
Public Const GS_MPEG_LANGAUGE_DUTCH As Long = &H20
Public Const GS_MPEG_LANGAUGE_DANISH As Long = &H40
Public Const GS_MPEG_LANGAUGE_FINNISH As Long = &H80
Public Const GS_MPEG_LANGAUGE_ITALIAN As Long = &H100
Public Const GS_MPEG_LANGAUGE_GREEK As Long = &H200
Public Const GS_MPEG_LANGAUGE_PORTUGUESE As Long = &H400
Public Const GS_MPEG_LANGAUGE_SWEDISH As Long = &H800
Public Const GS_MPEG_LANGAUGE_RUSSIAN As Long = &H1000
Public Const GS_MPEG_LANGAUGE_CHINESE As Long = &H2000
Public Const gsMpegCCFormat As Long = gsMpegLanguageCode + 1
Public Const GS_MPEG_CC_FORMAT_CCUBE As Long = &H1
Public Const GS_MPEG_CC_FORMAT_ATSC As Long = &H2
Public Const GS_MPEG_CC_FORMAT_CCUBE_REORDER As Long = &H4
```

```

Public Const GS_MPEG_CC_FORMAT_ATSC_REORDER As Long = &H8
Public Const gsMpegConcealmentVector As Long = gsMpegCCFormat + 1
Public Const gsMpegClosedGop As Long = gsMpegConcealmentVector + 1
Public Const gsMpegAdjustGopTC As Long = gsMpegClosedGop + 1
Public Const gsMpegAltCoEffTable As Long = gsMpegAdjustGopTC + 1
Public Const gsMpegNonLinearQuant As Long = gsMpegAltCoEffTable +
1
Public Const gsMpegMuxRate As Long = gsMpegNonLinearQuant + 1
Public Const gsMpegAudPacketSize As Long = gsMpegMuxRate + 1
Public Const gsMpegVidPacketSize As Long = gsMpegAudPacketSize + 1
Public Const gsMpegAudioStreamID As Long = gsMpegVidPacketSize + 1
Public Const gsMpegVideoStreamID As Long = gsMpegAudioStreamID + 1
Public Const gsMpegAudioStreamPID As Long = gsMpegVideoStreamID +
1
Public Const gsMpegVideoStreamPID As Long = gsMpegAudioStreamPID
+ 1

```

' Signal and Compression

```

Public Const gsSignalFormat As Long = 900
Public Const GS_SIGFORM_NTSC As Long = &H1
Public Const GS_SIGFORM_PAL As Long = &H2
Public Const GS_SIGFORM_CCIR_NTSC As Long = &H4
Public Const GS_SIGFORM_CCIR_PAL As Long = &H8
Public Const GS_SIGFORM_1035i_30_260M As Long = &H100
Public Const GS_SIGFORM_1035i_30X_260M As Long = &H200
Public Const GS_SIGFORM_1080i_30 As Long = &H400 /* (274M-
1997 Table1 System 4) */
Public Const GS_SIGFORM_1080i_30X As Long = &H800 /* (274M-
1997 Table1 System 5) */
Public Const GS_SIGFORM_1080i_25 As Long = &H1000 /* (274M-
1997 Table1 System 6) */
Public Const GS_SIGFORM_1080i_24 As Long = &H2000
Public Const GS_SIGFORM_1080i_24X As Long = &H4000
Public Const GS_SIGFORM_1080_30 As Long = &H800 /* (274M-
1997 Table1 System 7) */
Public Const GS_SIGFORM_1080_30X As Long = &H10000 /* (274M-
1997 Table1 System 8) */
Public Const GS_SIGFORM_1080_25 As Long = &H20000 /* (274M-
1997 Table1 System 9) */
Public Const GS_SIGFORM_1080_24 As Long = &H40000 /* (274M-
1997 Table1 System 10) */
Public Const GS_SIGFORM_1080_24X As Long = &H80000 /* (274M-
1997 Table1 System 11) */
Public Const GS_SIGFORM_720_60 As Long = &H100000 /*
(296M-1996 Table1 System 1) */
Public Const GS_SIGFORM_720_60X As Long = &H200000 /*
(296M-1996 Table1 System 2) */

```

'----

```

Public Const gsCompType As Long = gsSignalFormat + 1
Public Const GS_COMPTYPE_SOFTWARE As Long = &H1
Public Const GS_COMPTYPE_MJPEG As Long = &H2
Public Const GS_COMPTYPE_WAVELET As Long = &H8
Public Const GS_COMPTYPE_MPEG1 As Long = &H10
Public Const GS_COMPTYPE_MPEG2 As Long = &H20
Public Const GS_COMPTYPE_MPEG2I As Long = &H40
Public Const GS_COMPTYPE_MPEG2IBP As Long = &H80
Public Const GS_COMPTYPE_DV25 As Long = &H100
Public Const GS_COMPTYPE_DV50 As Long = &H200
Public Const GS_COMPTYPE_DVSD As Long = &H400
Public Const GS_COMPTYPE_DV100 As Long = &H800
Public Const GS_COMPTYPE_UN8BIT As Long = &H1000
Public Const GS_COMPTYPE_UN10BIT As Long = &H2000
Public Const GS_COMPTYPE_HDPAN As Long = &H10000
Public Const GS_COMPTYPE_HDSONY As Long = &H20000
Public Const GS_COMPTYPE_HDUNCOMP As Long = &H20000
'---
Public Const gsCompRateSize As Long = gsCompType + 1
Public Const gsCompRateRatio As Long = gsCompRateSize + 1
Public Const gsCompRatePercent As Long = gsCompRateRatio + 1
Public Const gsCompGOPSize As Long = gsCompRatePercent + 1
Public Const gsCompIFactor As Long = gsCompGOPSize + 1
Public Const gsCompBFactor As Long = gsCompIFactor + 1
Public Const gsCompPFactor As Long = gsCompBFactor + 1
Public Const gsCompRefPeriod As Long = gsCompPFactor + 1
Public Const gsTotalStorageAvail As Long = gsCompRefPeriod + 1
Public Const gsTotalStorageFree As Long = gsTotalStorageAvail + 1
Public Const gsTotalTimeAvail As Long = gsTotalStorageFree + 1
Public Const gsTotalTimeFree As Long = gsTotalTimeAvail + 1
Public Const gsVtrType As Long = gsTotalTimeFree + 1

' Other - Needs sub types
Public Const gsLocal As Long = 1000
Public Const gsSupportedFileTypes = gsLocal + 1
Public Const GS_SUPFILE_AVI As Long = &H1
Public Const GS_SUPFILE_ODML As Long = &H2
Public Const GS_SUPFILE_QT As Long = &H4
Public Const GS_SUPFILE_OMFI As Long = &H8
Public Const GS_SUPFILE_FIX As Long = &H100
Public Const GS_SUPFILE_AUDONLY As Long = &H10000
Public Const GS_SUPFILE_STILLS As Long = &H100000
Public Const GS_SUPFILE_UNK As Long = &H40000000
Public Const GS_SUPFILE_ANY As Long = &H80000000
Public Const gsIgnoreFileTypes = gsSupportedFileTypes + 1
Public Const gsRecInhibit As Long = gsIgnoreFileTypes + 1 '
Inhibit recording
Public Const gsRecDrive As Long = gsRecInhibit + 1
Public Const gsRecFileName As Long = gsRecDrive + 1

```

```

Public Const gsRecRate As Long = gsRecFileName + 1
Public Const gsRecFileFormat As Long = gsRecRate + 1
Public Const gsRecAudFileFormat As Long = gsRecFileFormat + 1
Public Const gsInsInhibit As Long = gsRecAudFileFormat + 1
Public Const gsDelInhibit As Long = gsInsInhibit + 1
Public Const gsConvertFileFormat As Long = gsDelInhibit + 1
Public Const gsConvertAudFileFormat As Long = gsConvertFileFormat
+ 1
Public Const gsDefStillLen As Long = gsConvertAudFileFormat + 1
Public Const gsSysTime As Long = gsDefStillLen + 1 'System
Reference time
Public Const gsDSyncMs As Long = gsSysTime + 1
Public Const gsHwPort As Long = gsDSyncMs + 1 ' Comport or
whatever
Public Const gsPBEE As Long = gsHwPort + 1 ' dwPosition
Public Const GS_PBEE_AUTO As Long = &H0
Public Const GS_PBEE_PB As Long = &H1
Public Const GS_PBEE_DEFAULT As Long = &HFF
Public Const gsServoRefSelect As Long = gsPBEE + 1 ' dwPosition
Public Const GS_SERVOREF_AUTO As Long = &H0
Public Const GS_SERVOREF_EXT As Long = &H1
Public Const GS_SERVOREF_DEFAULT As Long = &HFF
Public Const gsHeadSelect As Long = gsServoRefSelect + 1
'dwPosition
Public Const GS_HEADSEL_RECPLAY As Long = &H0
Public Const GS_HEADSEL_PLAY As Long = &H1
Public Const GS_HEADSEL_DEFAULT As Long = &HFF
Public Const gsColorFrame As Long = gsHeadSelect + 1 ' dwPosition
Public Const GS_CLRFRM_2FLD As Long = &H0
Public Const GS_CLRFRM_4FLD As Long = &H1
Public Const GS_CLRFRM_8FLD As Long = &H2
Public Const GS_CLRFRM_DEFAULT As Long = &HFF
Public Const gsVidRefDisable As Long = gsColorFrame + 1
'dwPosition
Public Const GS_VIDREF_DISABLE As Long = &H0
Public Const GS_VIDREF_ENABLE As Long = &H1

' Version
Public Const gsVWVersion As Long = 60000
Public Const gsMEVersion As Long = gsVWVersion + 1
Public Const gsVWType As Long = gsMEVersion + 1
Public Const gsVWChannelType As Long = gsVWType + 1

' HWND
Public Const gsMonitor As Long = 64000
Public Const gsHwnds As Long = gsMonitor + 1

'
Public Const gsChannelsExist As Long = 65536

```

```
Public Const gsClipMode As Long = gsChannelsExist + 1
Public Const gsRecOffset As Long = gsClipMode + 1
Public Const gsChanCapabilities As Long = gsRecOffset + 1
Public Const GS_CHANCAP_MPEG_ENC As Long = &H40000000
Public Const gsLastChangeMs As Long = gsChanCapabilities + 1
Public Const gsGPIIn As Long = gsLastChangeMs + 1
Public Const gsGPIOut As Long = gsGPIIn + 1

Public Const gsSaveCurrent As Long = 100000

' Global Returns
Public Const GS_NOT_SUPPORTED As Long = &HFFFFFFF
Public Const GS_BAD_PARAM As Long = &HFFFFFFE
```