MPEGIO2 SDK User Manual

Writing Software for Dual-Channel, Multi I/O, Colour Overlay & MPEG2/4/1 Encoder+Decoder PCIe Card

Version 1.0.2

Table of Contents

1. Introduction

MPEGIO2 SDK (Software Development Kit) is for writing customized application software to control **MPEGIO2 ---** a PCI-Express card with two independent channels each is capable of multi-socket Video/Audio I/O and mixing, colour text/graphics overlay, hardware MPEG encoding / decoding, real-time digital-I/O control and many other powerful functions. **MPEGIO2 SDK** shields the application developer from interfacing the complicated hardware registers on board the PCB, or programming the DirectShow filters directly, by using simple function calls to accomplish complicated tasks, such as multi-channel video preview or colour graphics overlay on MPEG video. Complete C++ source code with Microsoft VisualStudio projects, plus MS Windows **.Net** application class library and VB, C++.net, C#.net and VB.net samples with source codes help developers quickly and easily write powerful application software to control the hardware for real-time video preview, graphics overlay, MPEG encoding/decoding/streaming, digital I/O and many other operations.

The main functions and capabilities of the **MPEGIO2** card include:

--- Real-time encode MPEG2, MPEG4 and MPEG1 video using on-board hardware compression IC

--- Real-time decode MPEG2/4/1 video onto PC screen and external TV monitors simultaneously

--- 2 Independent Encoding+Decoding Channels per PCIe Card, up to 16 channels per PC allowed

--- Real-time Encode from 128Kbps Mbps up to **15 Mbps** MPEG Video Streams per Channel with Audio

--- Overlay Colour Graphics & Text Simultaneously on Encoded MPEG Video and on external TV Screen

--- Multiple input video as picture-in-picture, picture-by-picture are previewed, encoded and streamed

--- Overlay multiple colour text, graphics, timer/counter, transparency boxes on the encoded MPEG video

--- Real-time live multi-channel preview video on PC screen in re-sizable/ movable window or full screen

--- Simultaneous Video Input/Output per Channel on 2xRCA, 1xSVideo In **+** 2xRCA/SVideo Out Sockets

--- 2 Stereo Audio Input & Output Sockets per Channel, Input selectable between Line-in and Stereo Mic.

--- 4 Digital I/O Sockets per Channel for real-time control to external devices, digital input is via interrupt

--- Perfect Audio/Video Synchronization maintained on previewed, recorded and streamed MPEG video

--- Recorded MPEG files can seamlessly manually or automatically split at fixed time or length in real-time

--- Record Video using timer or calendar scheduler with daily or weekly repeat options

--- Real-time stream video over IP network multi-cast or uni-cast independent of file recording status

--- Real-time flip input video horizontally or vertically and enlarge video from any point (2-times zoom)

--- PAL and NTSC encoding at various sizes from720X576, 720X480, down to 176X144, 176X120-Pixels

--- Either Program Stream (PS) or Transport Stream (TS) MPEG Video can be Encoded or Decoded

--- User can set encoding parameters inc. bit-rate, frame rate, frame size, GOP structure, sampling rate, etc.

--- Various Encoding Aspect-ratio supported inc. 4:3, 16:9, square pels and 2.21:1

--- Overlay text, timer, graphics, rectangle and box, with 252 colours and font, alpha and blink parameters

--- Each channel allows 4 video sub-windows for simultaneous live video in different sizes and positions

--- Sub-Windows have border, background, colour brightness, contrast, hue, saturation, sharpness control

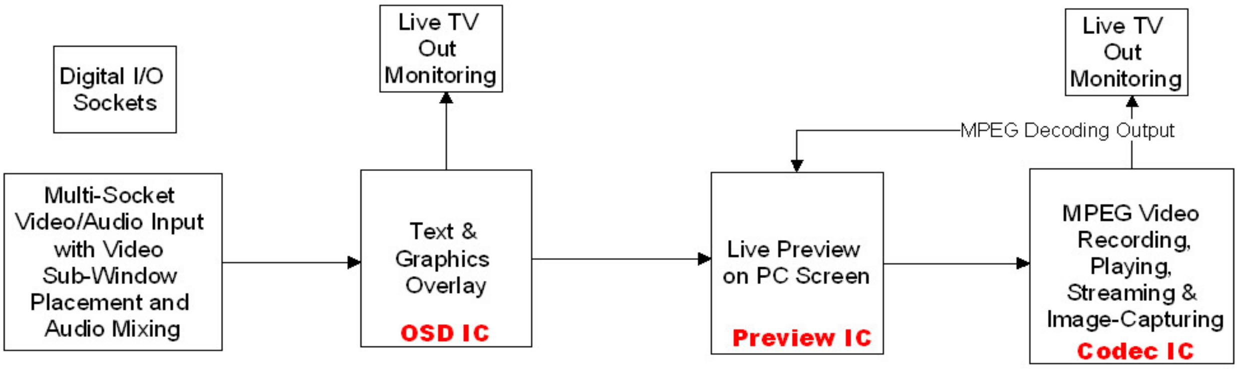
--- Each audio socket has hardware gain control and left/right mute, audio from input sockets can be mixed

--- Still images can be captured in real-time in bmp, jpg, gif, tif, and png format at 720X576/480-Pixel

--- Live recording status can be displayed inside video frame with user-definable colour, font and position

--- Device Driver, Application Software with Source Code and SDK for MS Windows XP and Windows 7

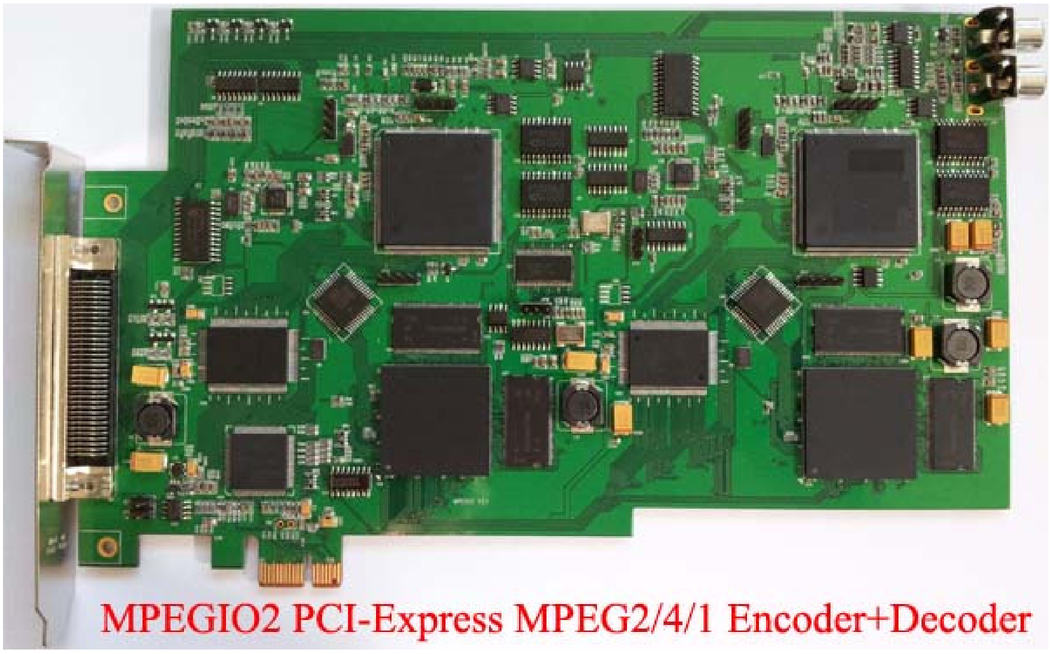
2. Functional and Input/Output Diagrams



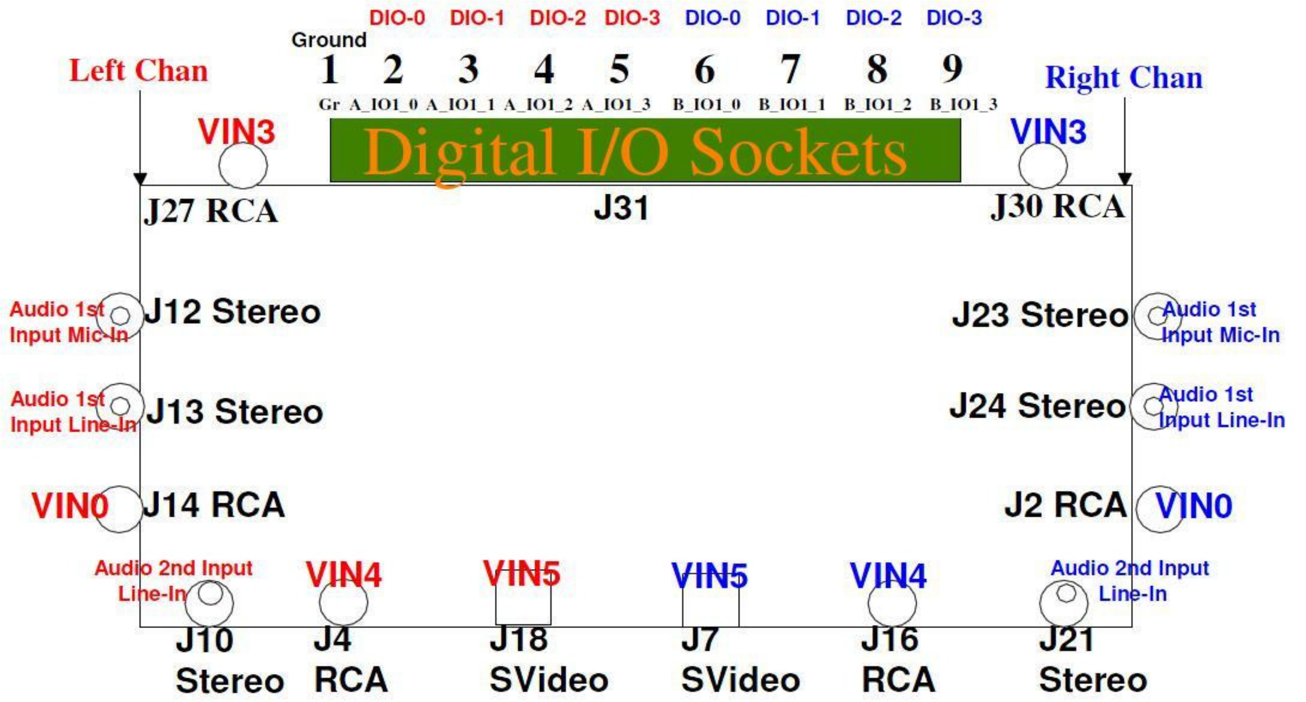
MPEGIO2 per Channel Functions

Each **MPEGIO2** card has two channels with identical functions operating independent of each other -- they can also route their video and/or audio output towards the matching channel's input (the name "**matching channel**" indicates the other channel on the same **MPEGIO2** card). The three major chips **OSD IC**, **Preview IC** and **Codec IC** of each channel can be individually hardware reset via SDK functions, and are responsible for Graphics Overlay, Video Preview and MPEG Encoding/Decoding/Streaming respectively.

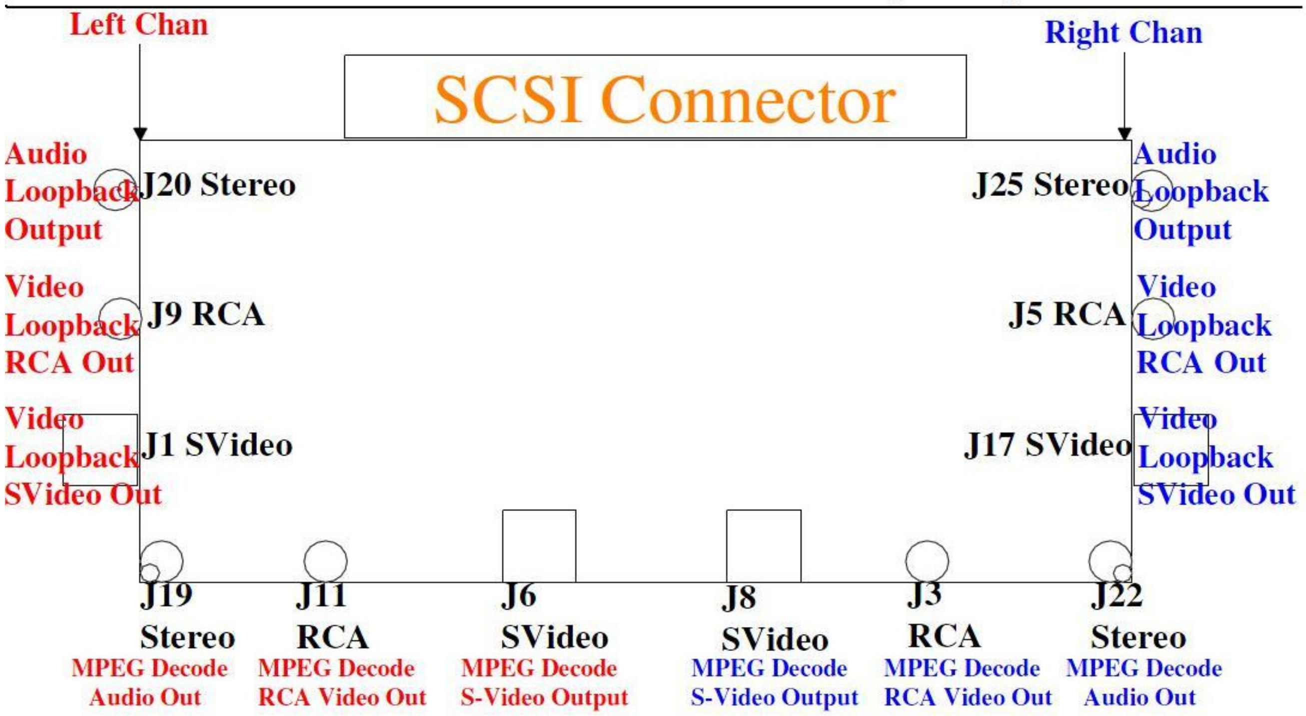
MPEGIO2 input and output video/audio via a two-layer **Breakout Box**, connected through a 68-pin SCSI cable, with identical female sockets on the back of the **MPEGIO2** card and on the **Breakout Box**:



See below for the **Breakout Box**'s top and bottom layer socket connections: the **Red** labels indicate the left(1st) channel sockets, the **Blue** labels indicate the right(2nd) channel sockets on the same card:



MPEGIO2 Breakout Box Top Layer



MPEGIO2 Breakout Box Bottom Layer

These sockets' detailed descriptions are in the sections of **Video I/O Functions**, **Audio I/O Functions** and **Digital I/O Functions**. It is obvious to see that the **Top Layer** contains Video/Audio **Input** Sockets plus Digital I/O sockets, while the **Bottom Layer** contains Video/Audio **Output** Sockets.

The **Black** labels **J1**, **J2, AIO1_0** etc. are on the PCB board, useful for identifying each socket.

3. MPEGIO2 SDK Components

MPEGIO2 SDK is based on a dynamic linking library **MPEGIO2.dll** which supports all the function calls application software might use, and several supporting libraries: "vwSDKR.dll", "UdpSender.dll", "MulticastSender.dll", etc.: **MPEGIO2.dll** and these supporting libraries need to be copied onto the PC before the application software can use the SDK. To facilitate MS Windows **.Net** environment programming, an **MPEGIO2API.dll .Net Class Library** is also included with source codes that wraps most functions from **MPEGIO2.dll** (users can easily add more using the C# source code & VS Project).

4. SDK Function Details

Note 1: "Output Parameters" in any of the following functions will only be set properly when those function calls succeed. If a function call failed, Output Parameters' values will be uncertain.

Note 2: All variable types have these **width** definitions**:**

long, int, LONG: signed 32-bit integer;
unsigned long, unsigned int, bool, ULONG, DWORD, COLORREF: unsigned 32-bit integer;
short: signed 16-bit integer;
unsigned short, WORD: unsigned 16-bit integer;
char: signed 8-bit integer;
BYTE, byte: unsigned 8-bit integer;
All pointers (char *, short *, int *, long *, bool *, etc): unsigned 32-bit memory address(integer).

Note 3: MPEGIO2 SDK runs on 32-bit MS Windows XP, Windows 7 and above, 64-bit operating systems are not supported.

4.1 SDK Start and Stop Functions

```
MPEGIO2_API unsigned long     MPEGIO2_initSDK(
                              HWND parentWnd,
                              unsigned long &totalMonitors,
                              bool &hasUnmatchedDrv,
                              unsigned short *MPEGEncoderType = NULL,
                              bool useInitFile = true,
                              bool redrawOverlays = true,
                              unsigned long NoCodecIC = 0,
                              bool verifyFW = true,
                              bool loadOSDRegs = true,
                              char *OSDRegsFile = 0,
                              bool load5ThRegs = true,
                              char *fifthInputRegsFile = 0,
                              bool loadOutputRegs = true,
                              char *outputRegsFile = 0,
                              bool noSignalLoadPAL = true,
                              unsigned long extra = 0);
```

Function: This is usually the first function to call when using the SDK: initializes all **MPEGIO2** channels and returns **totalChans** (this will be used in all following function descriptions to indicate the total channel number): the total number of **MPEGIO2** channels with device drivers installed properly on the PC. On failure this function returns 0. Most **MPEGIO2 SDK** functions can only work after **MPEGIO2_initSDK** returns **totalChans** between 1 and 16 (16 is the maximum **MPEGIO2** channels allowed per PC).

Input Parameter **parentWnd**: Handle of the application software created window passed to the **MPEGIO2** SDK for the application to receive messages (see "**SDK to Application Messages**" section) generated by the SDK. If this is not a valid window then no message will be passed from the SDK to application.

Output parameter **totalMonitors**: The total Monitors available on this PC's graphics cards.

Output parameter **hasUnmatchedDrv**: If this is returned true then there are some **MPEGIO2** Video Preview IC device

Drivers (**Inventa MPEGIO2 SAA7134)** that do not have their corresponding MPEG Codec IC driver (**Inventa MPEGIO2 VW2010)** installed,
or there are some **MPEGIO2** MPEG Codec IC device drivers that do not have their corresponding Video Preview IC device drivers installed.
When this happens, it's recommended (although not absolutely necessary) to exit the application and rectify those missing device drivers (e.g. install them) before re-start the **MPEGIO2** SDK.

Input parameter **MPEGEncoderType**: If not NULL must point to an array of 16 unsigned short (16-bit integer) members, of which each i'th member (**i** is 0 ~ 15) is either 0 (Program Stream) or 1 (Transport Stream) indicating the i'th **MPEGIO2** channel's MPEG encoder type.
If this parameter is NULL then each **MPEGIO2** channel's **encoder type** will be intialized to be the type used when **MPEGIO2** SDK exit last time as recorded in MPEGIO2.INI file.
If this is NULL and there is no MPEGIO2.INI file (such as when the very first time **MPEGIO2** SDK was used on a PC), Program Stream encoder type (0) will be used for all channels.
Note 1: the **encoder type** can be later changed by calling function **MPEGIO2_newEncoderStreamType**().
Note 2: the **encoder type** will decide the MPEG stream type to be encoded and decoded, i.e.,
Program Stream encoder can only encode and decode Program Stream MPEG files/streams,
Transport Stream encoder can only encode and decode Transport Stream MPEG files/streams.

Input parameter **useInitFile**: If true, device configuration values are read from file "softwareName.ini" (default to "**MPEGIO2.ini**", i.e., softwareName" defaults to "MPEGIO2" unless it is changed by calling **MPEGIO2_setSoftwareName**()): this file will be referred to as the "**initFile"** in all of the following SDK function descriptions. If false, device configuration values described in section "**Default Parameter Values**" of the "**MPEGIO2 Application Software Manual**" will be used.

Input parameter **redrawOverlays**: If true, the first call to **MPEGIO2_startPreviewWindow** after SDK is initialized through **MPEGIO2_initSDK**() will redraw all existing Overlay items read back from the MPEGIO2.ini file. Note if **MPEGIO2_startPreviewWindow** is never called then regardless this parameter's value, existing overlays will **not** be redrawn by the SDK, in that case the application can call **MPEGIO2_redrawAllOverlays** to redraw Overlays.

Input parameter **NoCodecIC**: Default is zero. If not 0, each binary bit 1 in the lowest 16 bits of this variable (starting from LSB) represents not using the MPEG Codec IC of that **MPEGIO2** channel number(bit 0 for channel 0, bit 1 for channel 1, etc.): in this way video preview, OSD etc. operation can still operate on that channel but no MPEG encoding/decoding/streaming operation is allowed on that channel.
Disabling any Codec IC will speed up the SDK loading since creating Codec IC structure is time consuming, therefore setting this parameter can be useful for debugging.
Use function **MPEGIO2_hasMPEGCodecIC**() to test if a channel has its MPEG Codec IC enabled.

Input parameter **verifyFW**: If some channels' MPEG Codec IC will be operating, setting this parameter to false means not to verify the downloaded firmware to the MPEG Codec ICs: this will lead to slightly faster initialization of the Codec IC, although risking not to find a possibly faulty IC or faulty associated SDRAM on the PCIe card. Default is true meaning always verifying downloaded firmware.

Input parameter **loadOSDRegs**: True means loading each channel's **OSD IC**'s register values from the parameter **OSDRegsFile** if it's a valid file name inc. path, or from the default OSD Register file name (OSDValsPAL.ini or OSDValsNTSC.ini) under the same path of the **MPEGIO2.DLL** if the parameter **OSDRegsFile** is not a valid file name inc. path.
Note all current overlay items will be erased on each channel if register value loading happens from the default file or file pointed to by parameter "**OSDRegsFile**".
False means do not load any register value for the **OSD IC**. Default is True.
Note the **OSD IC** is also reponsible for processing signal input for video sources 1 ~ 4 (VIN0 ~ VIN3).

Input parameter **OSDRegsFile**: If not NULL and parameter **loadOSDRegs** is true then this parameter must point to the full path and file name of the register values for the **OSD IC**. Default is NULL.

Input parameter **load5ThRegs**: True means loading each channel's 5Th Input Processing IC's register values from the parameter **fifthInputRegsFile** if it's a valid file name inc. path, or from the default 5Th Input Processing IC's Register file name(Input5ValsPAL.ini or Input5ValsNTSC.ini) under the same path of the **MPEGIO2.DLL** if the parameter **fifthInputRegsFile** is not a valid file name inc. path.
False means do not load any register value for the 5th Input IC. Default is True.

Input parameter **fifthInputRegsFile**: If not NULL and parameter load5ThRegs is true then this must point to the full path and file name of the register values for the 5Th Input processing IC. Default is NULL.

Input parameter **loadOutputRegs**: True means loading each channel's Video Output Processing IC's register values from the parameter **outputRegsFile** if it's a valid file name inc. path, or from the default Output Processing IC's Register file name (OutputPAL.ini or OutputNTSC.ini) under the same path of the MPEGIO2.DLL if the parameter **outputRegsFile** is not a valid file name inc. path.
False means do not load any register value for the Video Output Processing IC. Default is True.

Input parameter **outputRegsFile**: If not NULL and parameter **loadOutputRegs** is true then this must point to the full path and file name of the register values for the Video Output processing ICs. Default is NULL.

Input parameter **noSignalLoadPAL**: Only used if a channel has no input video signal at all of its input video sources,
and if **loadOSDRegs** is true but **OSDRegsFile** is not a valid file name,
or if **load5ThRegs** is true but **fifthInputRegsFile** is not a valid file name,
or if **loadOutputRegs** is true but **outputRegsFile** is not a valid file name:
in these cases the default OSD Register file name and/or default 5Th Input Processing IC's Register file name and/or default Output Processing IC's Register file name will be used:
parameter "**noSignalLoadPAL**" will decide to use PAL(if **noSignalLoadPAL** is true) or NTSC (if **noSignalLoadPAL** is false) type of the default register files. This parameter default to be true.

Input parameter **extra**: Currently must always be 0.

Return the number of found **MPEGIO2** channels.

Note 1: Returning zero means no **MPEGIO2** channel or driver is installed.
Note 2: **MPEGIO2** SDK can only operate properly if at least one channel's drivers are installed.
Note 3: **MPEGIO2** SDK does not specifically support or disallow multiple processes (multiple programs) to simultaenously start multiple copies of the **MPEGIO2.DLL** library: if one application called **MPEGIO2_initSDK,** other applications or processes calling **MPEGIO2_initSDK** and other **MPEGIO2** SDK functions should be very careful not to simultaneously call non-sharable functions, such as video preview, audio preview(monitor), still image capture, MPEG encoding and decoding, otherwise system crash or hanging will happen. With careful planning and co-operation **MPEGIO2.DLL** can be run simultaneously among multiple processess to accomplish concurrent accessing to the same or different **MPEGIO2** channels.

MPEGIO2_API void **MPEGIO2_endSDK**(ULONG delayBeforeEnd = 2000);
Function: End MPEGIO2 SDK, this must be called before exiting application software which previously called **MPEGIO2_initSDK**()
Input Parameter **delayBeforeEnd**: time in milli seconds the function will sleep before exit, default is 2000 (2 sec.).
Note: On WinXP, avoid calling **MPEGIO2_initSDK / MPEGIO2_endSDK** when other Video Overlay program (VideoLan, MediaPlayer etc.) is displaying video: this could corrupt video overlay or fail to preview video.

4.2 Generic SDK Functions

MPEGIO2_API char * **MPEGIO2_getSDKVer**(void);
Function: return the **MPEGIO2** SDK's version as a string such as "10.10.10" etc
Note: This function can be called without first calling **MPEGIO2_initSDK.**

MPEGIO2_API bool **MPEGIO2_getPCBVer**(unsigned long chanNum, int &ver);
Function: return the PCB hardware version for an **MPEGIO2** Channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **ver**: ver=0 for 1st version PCB, ver=1 for 2nd version PCB, etc...
Returns false on failure inc. if the channel has not been initalized.
Note 1: This function must be called between calling **MPEGIO2_initSDK**() and **MPEGIO2_endSDK**().
Note 2: The two channels on one **MPEGIO2** PCIe card always return the same PCB version value.
Note 3: Correct PCB ver. should be 3 or higher, a value 2 indicates switch IC failure, a value 1 indicates Input 5 IC failure.

MPEGIO2_API bool **MPEGIO2_getICBusDev**(unsigned long chanNum,
unsigned long *BusNum,
unsigned long *SAA7134DevNum,
unsigned long *VW2010DevNum);
Function: Get the on-board IC's Bus and Device Numbers for an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number.
Output parameter **BusNum**: If not NULL, return the PCI Bus Number for ICs on this **MPEGIO2** Channel.

Output parameter **SAA7134DevNum**: If not NULL, return the SAA7134 PCI Device Number.
Output parameter **VW2010DevNum**: If not NULL, return the VW2010 PCI Device Number.
Return True for success.

MPEGIO2_API void **MPEGIO2_setSoftwareName**(char *name);
Function: Assign user-defined program name to the SDK
Input parameter **name**: If not NULL, will be used as the program name that all message prompt/dialog display will use.
Note 1: This function can be called without calling **MPEGIO2_initSDK** first.
Note 2: If not set, the default program name will be "**MPEGIO2**".
Note 3: Maximum length of "name" is 100 bytes, contents exceeding 100 bytes will be ignored.

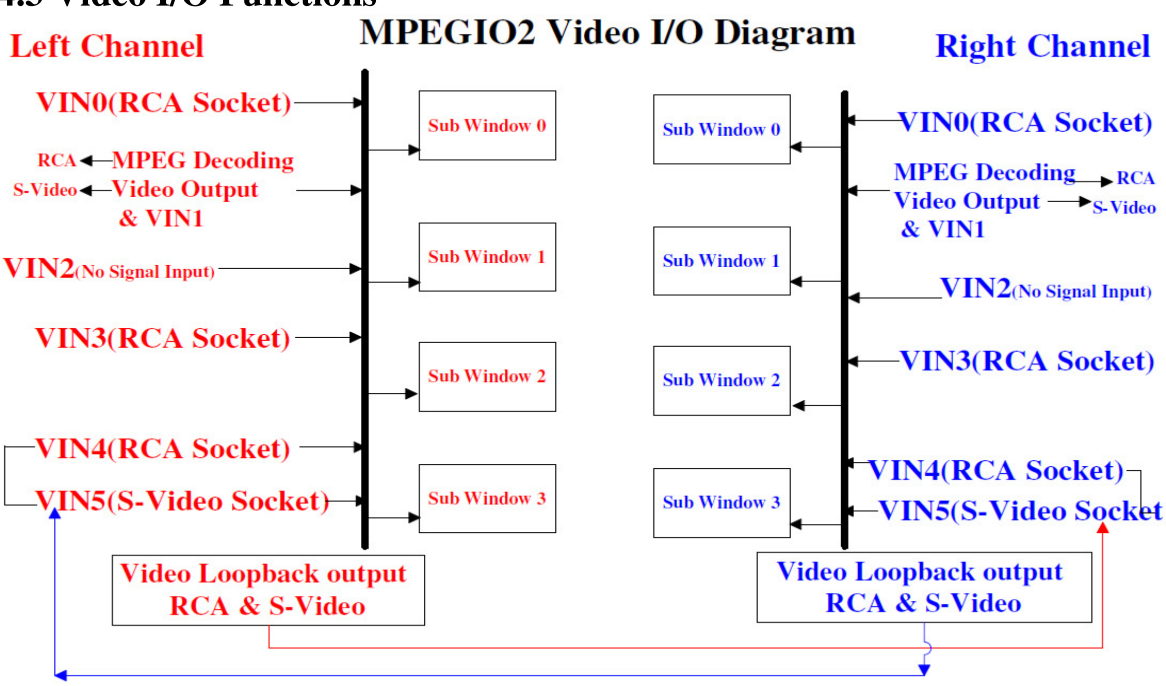
MPEGIO2_API bool **MPEGIO2_getSDKPath**(char **path);
Function: Return the full path as a null-terminated string where **MPEGIO2** SDK (**MPEGIO2.DLL**) resides.
Input Parameter **path**: if not null and this function returns true, this will point to the path (excluding the "**MPEGIO2.DLL**" name) where **MPEGIO2.DLL** resides.
Return true for success.
Note 1: This function can only be called after **MPEGIO2_initSDK** is called, regardless the existence of any **MPEGIO2** device driver, i.e., even if there is no **MPEGIO2** card installed this function can still return true.
Note 2: The application software should never modify the returned path string's value!!!

MPEGIO2_API unsigned long **MPEGIO2_getCPUMHz**(void);
Function: Return the PC's CPU clock frequency as MHz.
Note**:** This function can be called without calling **MPEGIO2_initSDK** first.

MPEGIO2_API bool **MPEGIO2_matchingChanNum**(unsigned long chanNum,
unsigned long &matchChanNum);
Function: From an **MPEGIO2** channel number, get its matching channel(on the same **MPEGIO2** card)'s channel number.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **matchChanNum**: If function returns true this will get the matching channel's number.
Return true for success. If this channel's matching channel does not exist, return false.

MPEGIO2_API bool **MPEGIO2_isLeftChan**(unsigned long chanNum);
Function: Return if the channel with number chanNum is the "left" channel(the 1st one) on the **MPEGIO2** card.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return true if the channel is the left one on the card, return false if the channel does not exist or is the "right" channel.
Note: Each **MPEGIO2** card has a left(1st) and a right(2nd) channel with I/O ports on the previous **Breakout Box** diagrams.

4.3 Video I/O Functions



Each **MPEGIO2** channel has **5 RCA** video **Input** sockets **VIN0, VIN1, VIN2, VIN3, VIN4**, **1 S-Video** input socket **VIN5**: **VIN4** and **VIN5** belong to the same "**5Th Input**" source (they are internally wired together, their input should come from the same source), while each of **VIN0~VIN3** can connect to a different video input source. On the **Breakout Box**, only **VIN0, VIN3, VIN4** and **VIN5** are visible, since **VIN1** is internally connected to the MPEG decoding (Playback) video output, and **VIN2** is left unconnected (always "no signal"). **MPEGIO2** uses 4 video "**Sub Windows**" (0 ~ 3) to show any video input on PC screen and encoded MPEG video stream: any of these 4 **Sub Windows** can use any of the 6 video input sockets (**VIN0~VIN5**) as its video source, as illustrated at the start of this section.

This "**4(Sub Windows), 5(Input Sources), 6(Input Sockets)**" naming convention will be used repeatedly in the following Video I/O related SDK functions.

Each **MPEGIO2** channel also has 2 pairs of RCA and S-Video **Output** sockets: one (Loopback Output) for displaying input and overlay, the other(MPEG Decode Out) for outputting MPEG decoding video.

4.3.1 Video Signal Functions

MPEGIO2_API bool **MPEGIO2_setVideoStandard**(unsigned long chanNum,
unsigned long standard);

Function: Set video standard for preview on PC screen and signal format on output sockets of an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **standard**: 1 for NTSC, 2 for PAL, 3 for SECAM, 0 for No Signal
Return true for success.

MPEGIO2_API unsigned long **MPEGIO2_getVideoStandard**(unsigned long chanNum);
Function: Get the video standard for preview on PC screen and signal format on output sockets of an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return 1 for NTSC, 2 for PAL, 3 for SECAM, 0 for No Signal or failure.

MPEGIO2_API int **MPEGIO2_videoInputSrcSignalType**(unsigned long chanNum, int inputNum);
Function: Return the video signal type of an input source.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **inputNum**: 0 ~ 5: Input source number, 0~3 for Input source VIN0~VIN3 of RCA, 4 is RCA of VIN4, 5 is S-Video of VIN5: each inputNum can be assigned to a sub-window by **MPEGIO2_setSubWinVideoSource**.
Return TV_NONE(0), TV_NTSC(1), or TV_PAL(2).
Note: VIN4's RCA and VIN5's S-Video are internally connected so their signal type are always same.

MPEGIO2_API unsigned short **MPEGIO2_videoInputActivelyUsed**(
unsigned long chanNum,
int inputNum,
unsigned short *subWnds = 0);

Function: Test if the input source "**inputNum**" is currently used as video source by some Enabled Sub Window.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputNum**: 0 ~ 5: which input source to test, 0~3 for Input source VIN0~VIN3 that are RCA socket only, 4 is CVBS(RCA) for VIN4, 5 is SVideo socket for VIN4, as selected by function **MPEGIO2_setSubWinVideoSource**.
Input Parameter **subWnds**: If not NULL, must point to a 4 member unsigned short integers to receive the Sub Window Numbers that are currently enabled and using **inputNum** as their video source, if this function returns N > 0: the first N members contain the enabled Sub Window Numbers using **inputNum** as video source.
Return the number of Enabled Sub Windows that are using the input source "inputNum" as their video source, return 0 for no such sub window or failure.

MPEGIO2_API int **MPEGIO2_firstSrcHasSig**(unsigned long chanNum, int &src);
Function: Return signal type TV_PAL(2) or TV_NTSC(1) of the first video input source that has TV signal and that is currently used by at least one Sub Window as its input souce.
Input Parameter **chanNum**: The **MPEGIO2** Channel Number, must be within 0 ~ totalChans - 1

Output Parameter **src**: the first input source (0~4) having signal and being used by some Sub Window as video input source.
When returning not TV_NONE, **src** indicates such an input source number:
0~3 means 1st, 2nd, 3rd, 4th input, 4 means the 5th input.
Return TV_PAL(2)/TV_NTSC(1) or TV_NONE(0). If any error (illegal chanNum etc.), also returns TV_NONE(0).
Note: If all Input Sources for all Sub Windows do not have signal, but some Input Sources not used by any Sub Window have signal this function still returns TV_NONE (0), because no Sub Window has input signal.

MPEGIO2_API bool **MPEGIO2_setOutputSignalType**(unsigned long chanNum,
unsigned long signalType,
bool defaultOutRegisters = true);

Function: Set up an **MPEGIO2** Channel's video signal type for PC screen/output TV port display and MPEG encoding.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **signalType**: Must be TV_PAL (2) or TV_NTSC(1)
Input Parameter **defaultOutRegisters**: If true(default), the 2 Video Output ICs responsible for MPEG decoding and Video Loopback Output on the Breakout Box will get loaded with their default register values according to "**signalType**".
Return True for success.

MPEGIO2_API bool **MPEGIO2_setNoVideoIndicateMode**(unsigned long chanNum, int mode);
Function: Set the indication mode for no-video sub-windows (when no signal is from the sub-windows' input source).
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **mode**: 0 ~ 3:
0 = Bypass (default): No action taken
1 = Capture last image
2 = Blanked with blank color
3 = Capture last image and blink channel boundary
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getNoVideoIndicateMode**(unsigned long chanNum, int &mode);
Function: Get the indication mode for no-video sub-windows (when no signal is from the sub-windows' input source).
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **mode**: 0 ~ 3: same as in function **MPEGIO2_setNoVideoIndicateMode**
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setNoChanIndicateMode**(unsigned long chanNum, int mode);
Function: Select the indication mode for no sub-window area
(areas within the video frame 720X576 or 720X480 pixels that are not covered by any of the 4 video sub-windows) using Y/Cb/Cr colour scheme.
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **mode**: 0 ~ 3:
In horizontal and vertical active region:
0: Background layer with background color (default) as set by function **MPEGIO2_setBkColour**
1: Y = 0, Cb/Cr = 128
2: Y/Cb/Cr = 0
3: Y/Cb/Cr = 0;

In horizontal and vertical blanking region:
0: Y = 16, Cb/Cr = 128 (default)
1: Background layer with background color
2: Y = 0, Cb = {0, F, V, 0, Cascade, linenum[8:7]}, Cr = {0, linenum[6:0]}
3: Y/Cb/Cr = 0
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getNoChanIndicateMode**(unsigned long chanNum, int &mode);
Function: Retrieve the indication mode for no sub-window area.
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output Parameter **mode**: 0 ~ 3: same meaning as in function "**MPEGIO2_setNoChanIndicateMode**"
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setOSDICSubCarrierFreq**(unsigned long chanNum,
unsigned char freq);

Function: Set the OSD IC's Video Output (Video Encoder) colour sub-carrier frequency
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **freq**: 0 ~ 3
0: 3.57954545 MHz (IC default)
1: 4.43361875 MHz (SDK default for PAL)
2: 3.57561149 MHz (SDK default for NTSC)
3: 3.58205625 MHz
Return true for success.
Note 1: See "http://countrycode.org/tv-standards" for Countries and their TV systems

Note 2: Video Input Formats Supported by the OSD IC:

Format	Lines	Fields	SubCarrierFrequency	Countries
NTSC	525	60	3.57954545 MHz	USA etc
NTSC-Japan	525	60	3.57954545 MHz	Japan
PAL-B, G, N	625	50	4.433619 MHz	Australia, New Zealand, etc.
PAL-D	625	50	4.433619 MHz	China
PAL-H	625	50	4.433619 MHz	Belgium
PAL-I	625	50	4.433619 MHz	UK, etc.
PAL-M	625	60	3.575612 MHz	Brazil
PAL-CN	625	50	3.582056 MHz	Argentina
SECAM	625	50	4.406 MHz	France, Eastern Europe,
			4.250 MHz	Middle East, Russia
PAL-60	525	60	4.433619 MHz	China
NTSC 4.43	525	60	4.433619 MHz	Transcoding

Note 3: Video Standard Selections:

Format	Line/Frequency(Hz)	Sub Caririer Frequency	Standard	Phase Alteration	Set 7.5IRE
NTSC-M	525/59.94	3.579545 MHz	60Hz	No	Yes
NTSC-Japan	525/59.94	3.579545 MHz	60Hz	No	No
NTSC-4.43	525/59.94	4.43361875 MHz	60Hz	No	Yes
NTSC-N	625/50	3.579545 MHz	50Hz	No	No
PAL-BDGHI	625/50	4.43361875 MHz	50Hz	Yes	No
PAL-N	625/50	4.43361875 MHz	50Hz	Yes	Yes
PAL-M	525/59.94	3.57561149 MHz	60Hz	Yes	No
PAL-NC	625/50	3.58205625 MHz	50Hz	Yes	No
PAL-60	525/59.94	4.43361875 MHz	60Hz	Yes	No

MPEGIO2_API bool **MPEGIO2_getOSDICSubCarrierFreq**(unsigned long chanNum,
unsigned char &freq);

Function: Get the OSD IC's Video Output (Video Encoder) colour sub-carrier frequency
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **freq**: Same as in function **MPEGIO2_setOSDICSubCarrierFreq**
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICPhaseAlteration**(unsigned long chanNum, bool enable);

Function: Enable/Disable the OSD IC's phase alternation for line-by-line.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **enable**:true to enable, false to disable phase alternation for line-by-line. Default: PAL:enable, NTSC:disable
Return true for success.

MPEGIO2_API bool **MPEGIO2_getOSDICPhaseAlteration**(unsigned long chanNum, bool &enable);

Function: Get the Enable/Disable status of the OSD IC's phase alternation for line-by-line.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **enable**: true to enable, false to disable phase alternation for line-by-line.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICPhaseAlterationReset**(unsigned long chanNum,
bool enable);

Function: Enable/Disable the OSD IC's Reset to the phase alternation every 8 field
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **enable**: true to enable, false (default) to disable OSD IC's Reset to the phase alternation every 8 field
Return true for success.

MPEGIO2_API bool **MPEGIO2_getOSDICPhaseAlterationReset**(unsigned long chanNum,
bool &enable);

Function: Get the Enable/Disable status of the OSD IC's Reset to the phase alternation every 8 field
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **enable**: true to enable, false (default) to disable OSD IC's Reset to the phase alternation every 8 field
Return true for success.

MPEGIO2_API bool **MPEGIO2_setHPLLMode**(unsigned long chanNum, unsigned long mode);

Function: Set Video Preview IC's Horizontal PLL Characteristics (horizontal sync and line locked clock generation)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **mode**: 0 = TV mode; for poor quality TV signals only, e.g. bad signal-to-noise ratio
1 = VTR mode; fast timing tracking for low noise input signals, but frequency deviations of output clock and sync signals are restricted;
3 = fast tracking and locking of HPLL (this is the default setting)
Any other mode value are illegal
Return true for success.

MPEGIO2_API unsigned long **MPEGIO2_getHPLLMode**(unsigned long chanNum);

Function: Get Video Preview IC's Horizontal PLL Characteristics (horizontal sync and line locked clock generation)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return: As parameter **mode** in function **MPEGIO2_setHPLLMode**
Any other return value inc. -1 means failure or input error.

MPEGIO2_API bool **MPEGIO2_setVSyncTrack**(unsigned long chanNum, unsigned long mode);

Function: Set Video Preview IC's Vertical Sync Tracking Mode
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **mode**:
0 (default) = normal operation; tracking the incoming video signal with some inertia,
surviving noisy signal and missing sync pulses
1 = fast tracking; applicable for stable sources, e.g. TV channel hopping
2 = free running; regular vertical sync output with nominal timing, not tracking the input signal (e.g. synthesize video timing)
3 = immediate mode; no inertia, e.g. immediately catching next vertical sync after source switch
Any other value is illegal.
Return true for success.

MPEGIO2_API unsigned long **MPEGIO2_getVSyncTrack**(unsigned long chanNum);

Function: Get Video Preview IC's Vertical Sync Tracking Mode
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return: Same as the parameter **mode** value in function **MPEGIO2_setVSyncTrack.**

4.3.2 Input Source Functions

MPEGIO2_API bool **MPEGIO2_setSubWinVideoSource**(unsigned long chanNum,
unsigned short subWinNum,
int src) ;

Function: Set Incoming Video Source on a Video Sub Window.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Input Parameter **src**: 1 for RCA socket of input 1 (VIN0-RCA, J14 or J2 on Breakout Box),
2 for RCA socket of input 2 (VIN1-RCA, MPEG Decode Output Port J11 or J3 on Breakout Box),
3 for RCA socket of input 3 (VIN2-RCA, Unconnected so is always "no signal"),
4 for RCA socket of input 4 (VIN3-RCA, J27 or J30 on Breakout Box),
5 for RCA socket on input 5(VIN4-RCA, J4 or J16 on Breakout Box),
6 for SVideo socket on input 5(VIN5-SVideo, J18 or J7 on Breakout Box).
Return true for success.
Note 1: Depending on the call to function **MPEGIO2_5ThInputSourceSet**(), the 5Th Input's video signal (**src** value 5 or 6) can be from either the 5Th input socket, or from this channel's matching channel's video loopback output.
Note 2: By default, Sub Window 0's video source is **src** 1(VIN0-RCA), Sub Window 1's video source is **src** 2(VIN1-RCA), Sub Window 2's video source is **src** 6(VIN5-SVideo), Sub Window 3's video source is **src** 4(VIN3-RCA).
Note 3: Multiple Sub Windows can have the same Video Source: e.g., SVideo (**src**=6) can be assigned to Sub Windows 1, 2.
Note 4: When **src** 6(SVideo) has signal input but **src** 5(RCA) doesn't and a Sub Window selects **src** 5 as its input source, or when **src** 5(RCA) has signal input but **src** 6(SVideo) doesn't and a Sub Window selects **src** 6 as its input source, the video displayed in that Sub Window will be black & white: because the **src** 5 is internally connected to **src** 6's Brightness pin.

MPEGIO2_API bool **MPEGIO2_getSubWinVideoSource**(
unsigned long chanNum,
unsigned short subWinNum,
int &src) ;

Function: Retrieve the Incoming Video Source Number currently set for a Video Sub Window.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Output Parameter **src**: 1 ~ 6, Same as **src** in function **MPEGIO2_setSubWinVideoSource.**
Return true for success.

MPEGIO2_API bool **MPEGIO2_5ThInputSourceSet**(ULONG chanNum, bool fromOtherChan);

Function: Select if the 5Th Video Input Source is from the matching channel on the same **MPEGIO2** Card
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **fromOtherChan**: True to use the video loopback output of the matching channel on the same **MPEGIO2** card as this channel's 5Th Video Input,
false (this is the default) to use this channel's RCA/SVideo sockets(of Input 5)' input signal as video souce for the 5Th Input.
Return: true for success.
Note 1: This setting is not automatically saved/restored by the SDK (i.e., on starting the **MPEGIO2** SDK each channel's 5Th Input Source is always set to RCA/SVideo sockets VIN4/VIN5 on the **Breakout Box**), so applications will need to save/restore this setting if they wish to automatically keep it in between application program start/stop.
Note 2: When connecting external video sources to the 5Th Video Input, always make sure the signals on sockets VIN4 RCA and VIN5 S-Video are from the same source (e.g., from the same VCR or camera), because internally these two sockets are connected, so using different external signals will result in corrupted signal display and encoded video.
Note 3: This function set Video Input source only. To set audio input source as from the matching channel, function **MPEGIO2_setAudioInLine** can be used.

MPEGIO2_API bool **MPEGIO2_5ThInputSourceGet**(ULONG chanNum, bool &fromOtherChan);

Function: Test if the 5Th Video Input Source is from the matching channel on the same **MPEGIO2** Card

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output Parameter **fromOtherChan**: True to use the video loopback output of the matching channel on the same **MPEGIO2** card as this channel's 5Th Video Input; false (default) to use this channel's RCA/SVideo sockets(VIN4/VIN5)' input signal as video souce for the 5Th Input.
Return: true for success.

4.3.3 Sub Window Functions

Note: For each sub window's video input source, see the previous **"Input Source Functions"** section.

```
MPEGIO2_API bool MPEGIO2_setSubWinPos( unsigned long chanNum,
                                       unsigned short subWinNum,
                                       unsigned short left,
                                       unsigned short top,
                                       unsigned short width,
                                       unsigned short height) ;
```

Function: Set up a Video Sub Window's position within the video frame.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Input Parameter **left**: the left position of the Sub Window, in pxiel unit
Input Parameter **top**: the top position of the Sub Window, in pxiel unit
Input Parameter **width**: the width of the Sub Window, in pxiel unit
Input Parameter **height**: the height of the Sub Window, in pxiel unit
Return True for success.
Note: For PAL video input, a Sub Window's position is within 0~719 horizontally and 0~575 vertically, for NTSC video input, the position is within 0~719 horizontally and 0~479 vertically. Any value set beyond these position limit will result in error.

```
MPEGIO2_API bool MPEGIO2_getSubWinPos(unsigned long chanNum,
                                      unsigned short subWinNum,
                                      unsigned short &left,
                                      unsigned short &top,
                                      unsigned short &width,
                                      unsigned short &height) ;
```

Function: Get a Video Sub Window's position within the video frame.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Output Parameter **left**: the left position of the Sub Window, in pxiel unit
Output Parameter **top**: the top position of the Sub Window, in pxiel unit
Output Parameter **width**: the width of the Sub Window, in pxiel unit
Output Parameter **height**: the height of the Sub Window, in pxiel unit
Return True for success.

```
MPEGIO2_API void MPEGIO2_setSubWinCommonPos(unsigned long chanNum, int commonPos);
```

Function: Set up a Video Sub Window's position using some pre-defined common position values
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **commonPos**: 1 ~ 19, defined as:

1 : Sub Window 0 occupies the full video frame in front of all other Sub Windows.
2 : Sub Window 1 occupies the full video frame in front of all other Sub Windows.
3 : Sub Window 2 occupies the full video frame in front of all other Sub Windows.
4 : Sub Window 3 occupies the full video frame in front of all other Sub Windows.
5 : Sub Window 0 ~ 3 each occupies a quarter of the video frame (equally split as 4 quarters):
Sub Window 0 at upper left quarter, Sub Window 1 at upper right quarter,
Sub Window 2 at lower left quarter, Sub Window 3 at lower right quarter.
6 : Sub Window 0 and Sub Window 1 occupy the left and right half of the full video frame respectively.
7 : Sub Window 0 and Sub Window 2 occupy the left and right half of the full video frame respectively.
8 : Sub Window 0 and Sub Window 3 occupy the left and right half of the full video frame respectively.
9 : Sub Window 1 and Sub Window 2 occupy the left and right half of the full video frame respectively.
10:Sub Window 1 and Sub Window 3 occupy the left and right half of the full video frame respectively.
11:Sub Window 2 and Sub Window 3 occupy the left and right half of the full video frame respectively.
12:Sub Window 0 and Sub Window 1 occupy the top and bottom half of the full video frame respectively.
13:Sub Window 0 and Sub Window 2 occupy the top and bottom half of the full video frame respectively.
14:Sub Window 0 and Sub Window 3 occupy the top and bottom half of the full video frame respectively.
15:Sub Window 1 and Sub Window 2 occupy the top and bottom half of the full video frame respectively.
16:Sub Window 1 and Sub Window 3 occupy the top and bottom half of the full video frame respectively.

17:Sub Window 2 and Sub Window 3 occupy the top and bottom half of the full video frame respectively.
18:Sub Window 0 & 3 as Picture-in-Picture with Sub Window 0 Full Frame,
Sub Window 3 occupies 1/6 Frame Size at Lower Right Corner.
19: Sub Window 2 & 0 as Picture-in-Picture with Sub Window 2 Full Frame,
Sub Window 0 occupies 1/6 Frame Size at Lower Left Corner.

MPEGIO2_API bool **MPEGIO2_setSubWinPopup**(unsigned long chanNum,
unsigned short subWinNum,
bool popup);

Function: Set a Video Sub Window's popup property that decides the display priority when two Sub Windows overlap.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Input Parameter **popup**: True to enable the popup of this Sub Window, false to disable it – when two Sub Windows overlap inside video frame, the overlapped area will appear or hide as follows:

- If the 2 Sub Windows have different "popup" property, video in the popup enabled Sub Window will appear
- If the 2 Sub Windows have the same "popup" property, video in the lower-numbered Sub Window will appear
- If one Sub Window is Disabled(See **MPEGIO2_setSubWinEnable**) the other Sub Window will appear
- If both Sub Windows are Disabled both will disappear (the area will have Background colour of this channel)

Note: by default, all Video Sub Windows have their "popup" property enabled, so by default, lower numbered Video Sub Window's content appear in front of the higher numbered Video Sub Windows if they overlap.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getSubWinPopup**(unsigned long chanNum,
unsigned short subWinNum);

Function: Get a Video Sub Window's popup property that decides the display priority when two Sub Windows overlap.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Return True means the popup of this Sub Window is enabled (default), false means it is disabled.

MPEGIO2_API bool **MPEGIO2_setSubWinEnable**(unsigned long chanNum,
unsigned short subWinNum,
bool enable);

Function: Enable or disable a Video Sub Window's capability to accept incoming video signal from its video source.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Input Parameter **enable**: True to enable, false to disable Sub Window – disallow signal from its video source to appear thus exposing the background colour or other enabled Sub Windows behind it. By default all Video Sub Windows are enabled.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getSubWinEnable**(unsigned long chanNum,
unsigned short subWinNum);

Function: Get the Enable or disable property of a Video Sub Window.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **subWinNum**: must be within 0 ~ 3, the number of the Video Sub Window
Return True if this Sub Window is enabled.

MPEGIO2_API bool **MPEGIO2_setSubWinMirror**(unsigned long chanNum,
unsigned short subWinNum,
bool HoriMirror,
bool VertMirror);

Function: Enable / disable video mirroring(flipping) effect on a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Input parameter **HoriMirror**: true to enable horizontal mirror of the video in the sub window, false to stop horizontal mirror
Input parameter **VertMirror**: true to enable vertical mirror, false to disable vertical mirror.
Return true if succeed.

MPEGIO2_API bool **MPEGIO2_getSubWinMirror**(unsigned long chanNum,
unsigned short subWinNum,

```
                                        bool &HoriMirror,
                                        bool &VertMirror);
```
Function: Get the video mirroring(flipping) enable/disable status on a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Input parameter **HoriMirror**: true to enable horizontal mirror of the video in the sub window, false to stop horizontal mirror
Input parameter **VertMirror**: true to enable vertical mirror, false to disable vertical mirror
Return true if succeed.

```
MPEGIO2_API bool MPEGIO2_setSubWinFreeze(  unsigned long chanNum,
                                           unsigned short subWinNum,
                                           bool freeze,
                                           bool imageEnhancement = true);
```
Function: Enable / disable video freezing on a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Input parameter **freeze**: true to freeze the sub window's video display, false to stop freezing the video display
Input parameter **imageEnhancement**: true to enable imageEnhancement, flase to disable image enhancement
Return true if succeed.

```
MPEGIO2_API bool MPEGIO2_getSubWinFreeze(  unsigned long chanNum,
                                           unsigned short subWinNum,
                                           bool &freeze);
```
Function: Get the video freezing status on a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Output parameter **freeze**: true means freezing the sub window's video display, false means not freezing the video display
Return true if succeed.

```
MPEGIO2_API bool MPEGIO2_setSubWinBoundary( unsigned long chanNum,
                                            unsigned short subWinNum,
                                            bool enable);
```
Function: Enable / disable boundary of a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Input parameter **enable**: true to enable the sub window's boundary, false to disable the sub window boundary
Return true if succeed.

```
MPEGIO2_API bool MPEGIO2_getSubWinBoundary( unsigned long chanNum,
                                            unsigned short subWinNum,
                                            bool &enable);
```
Function: Enable / disable boundary of a sub window within the video frame for an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Output parameter **enable**: true to enable the sub window's boundary, false to disable the sub window boundary
Return true if succeed.

```
MPEGIO2_API bool MPEGIO2_setSubWinBoundaryBlink(
                                            unsigned long chanNum,
                                            unsigned short subWinNum,
                                            bool enable);
```
Function: Enable / disable boundary blinking of a sub window within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Input parameter **enable**: true to enable the sub window's boundary blinking, false to disable boundary blinking.
Return true if succeed.

```
MPEGIO2_API bool  MPEGIO2_getSubWinBoundaryBlink(
                                                  unsigned long chanNum,
                                                  unsigned short subWinNum,
                                                  bool &enable);
```

Function: Get boundary blinking status of a sub window within the video frame for channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **subWinNum**: sub window number within the video frame: 0~3
Output parameter **enable**: true to enable the sub window's boundary blinking, false to disable blinking
Return true if succeed.

4.3.4 Video Colour Functions

MPEGIO2_API bool **MPEGIO2_setInputBright**(unsigned long chanNum,
unsigned short inputSrc,
char bright);

Function: Set an Input Source's video brightness.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5^{th} Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Input Parameter **bright**: If inputSrc is 1~4, the brightness values are within –128 to 127 in 2's complement format.
Larger values increase brightness. A value 0(default) has no effect on the data.
If **inputSrc** is 5, the brightness values are within 0 to 255 as unsigned binary values, default is 128.
Larger value increases brightness. Darkest value is 0, brightest value is 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputBright**(unsigned long chanNum,
unsigned short inputSrc,
char &bright);

Function: Get an Input Source's video brightness.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5^{th} Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Output Parameter **bright**: If inputSrc is 1~4, the brightness values are within –128 to 127 in 2's complement format.
Larger values increase brightness. A value 0(default) has no effect on the data.
If inputSrc is 5, the brightness values are within 0 to 255 as unsigned binary values, default is 128.
Larger value increases brightness. Darkest value is 0, brightest value is 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setInputContrast** (unsigned long chanNum,
unsigned short inputSrc,
unsigned char contrast);

Function: Set an Input Source's video contrast.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5^{th} Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Input Parameter **contrast**: the Input Source's contrast, within 0 ~ 255. When **inputSrc** is 1~4, default is 100(hex 64), when
inputSrc is 5, default is 128.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputContrast** (unsigned long chanNum,
unsigned short inputSrc,
unsigned char &contrast);

Function: Get an Input Source's video contrast.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5^{th} Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Output Parameter **contrast**: the Input Source's contrast, within 0 ~ 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setInputHue**(unsigned long chanNum,
unsigned short inputSrc,
char hue);

Function: Set an Input Source's video hue.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:

1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Input Parameter **hue**: the Input Source's hue, within 0x00 ~ 0x80 as 2's complement number from
+36 degree (7Fh) to -36 degree (80h) with an increment of 0.28 degree.
Positive value gives greenish tone and negative value gives purplish tone. default is 0x00.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputHue**(unsigned long chanNum,
unsigned short inputSrc,
char &hue);

Function: Get an Input Source's video hue.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Output Parameter **hue**: the Input Source's hue, within 0x00 ~ 0x80 as 2's complement number from
+36 degree (7Fh) to -36 degree (80h) with an increment of 0.28 degree.
Positive value gives greenish tone and negative value gives purplish tone. default is 0x00.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setInputSharp**(unsigned long chanNum,
unsigned short inputSrc,
char sharp);

Function: Set an Input Source's video sharpness.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Input Parameter **sharp**: the Input Source's sharpness, within 0 ~ 15, default is 1.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputSharp**(unsigned long chanNum,
unsigned short inputSrc,
char &sharp);

Function: Get an Input Source's video sharpness.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Output Parameter **sharp**: the Input Source's sharpness, within 0 ~ 15.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setInputUsa** (unsigned long chanNum,
unsigned short inputSrc,
unsigned char usa);

Function: Set an Input Source's video U(Cb) Component Saturation or just Colour Saturaion.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Input Parameter **usa**: For input source 1~4, the Input Source's U colour component saturation, within 0 ~ 255, default is 128.
For input source 5, the Input Source's colour saturation, within 0 ~ 255, default is 128.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputUsa** (unsigned long chanNum,
unsigned short inputSrc,
unsigned char &usa);

Function: Get an Input Source's video U(Cb) Component Saturation or just Colour Saturaion.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~5 indicating one of the 5 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets,
5 for the 5th Input Source (on VIN4 RCA and VIN5 S-Video sockets)
Output Parameter **usa**: For input source 1~4, the Input Source's U colour component saturation, within 0 ~ 255.
For input source 5, the Input Source's colour saturation, within 0 ~ 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setInputVsa (**unsigned long chanNum,
unsigned short inputSrc,
unsigned char vsa);
Function: Set an Input Source's video V(Cr) Component Saturation or just Colour Saturaion.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~4 indicating one of the 4 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets.
Note the 5th input source has no Vsa.
Input Parameter **vsa**: the Input Source's V colour component saturation, within 0 ~ 255, default is 128.
Return True for success.

MPEGIO2_API bool **MPEGIO2_getInputVsa (**unsigned long chanNum,
unsigned short inputSrc,
unsigned char &vsa);
Function: Get an Input Source's video V(Cr) Component Saturation or just Colour Saturaion.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc**: 1~4 indicating one of the 4 video input sources:
1~4 for Input source VIN0~VIN3 of RCA sockets.
Note the 5th input source has no Vsa.
Output Parameter **vsa**: the Input Source's V colour component saturation, within 0 ~ 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setBright**(unsigned long chanNum, unsigned char bright);

Function: Set the Preview and MPEG Encoding Video Luminance Brightness on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **bright**: the new brightness value, within 0 ~ 255, default is 0x80:
0xFF 255 (bright)
0x80 128 (ITU level nominal)
0x00 0 (dark)
Return True for success.
Note: Brightness set by **MPEGIO2_setBright** affetcs overall video on screen and on encoded MPEG stream, but does not affect the Brightness on video loopback output ports Video Loopback RCA Out and Video Loopback S-Video Out (as labelled on the **Breakout Box**), while **MPEGIO2_setInputBright** does affect video Loopback RCA/S-Video Out, although only within one single input source.

MPEGIO2_API bool **MPEGIO2_getBright**(unsigned long chanNum, unsigned char &bright);
Function: Get the Preview and MPEG Encoding Video Luminance Brightness on an **MPEGIO2** Channel.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **bright**: the current brightness value, within 0 ~ 255, default is 0x80:
0xFF 255 (bright)
0x80 128 (ITU level nominal)
0x00 0 (dark)
Return True for success.

MPEGIO2_API bool **MPEGIO2_setContrast**(unsigned long chanNum, unsigned char contrast);
Function: Set the Preview and MPEG Encoding Video Luminance Contrast on an **MPEGIO2** Channel.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **contrast**: the new contrast value, within 0 ~ 255, default is 0x44:
0x7F 1.984 (maximum)

0x44 1.063 (ITU level) nominal
0x40 1.0
0x00 0 (luminance off), results to flat medium grey, if brightness set nominal
...
0xC0 -1.0 (inverse luminance)
0x80 -2.0 (inverse luminance) .

Return True for success.

Note: Contrast set by **MPEGIO2_setContrast** affetcs overall video on screen and on encoded MPEG stream, but does not affect the Contrast on video loopback output ports Video Loopback RCA Out and Video Loopback S-Video Out (as labelled on the **Breakout Box**), while **MPEGIO2_setInputContrast** does affect video Loopback RCA/S-Video Out, although only within one single input source.

MPEGIO2_API bool **MPEGIO2_getContrast**(unsigned long chanNum, unsigned char &contrast);

Function: Get the Preview and MPEG Encoding Video Luminance Contrast on an **MPEGIO2** Channel.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **contrast**: the current contrast value, within 0 ~ 255, default is 0x44:

0x7F 1.984 (maximum)
0x44 1.063 (ITU level) nominal
0x40 1.0
0x00 0 (luminance off), results to flat medium grey, if brightness set nominal
...
0xC0 -1.0 (inverse luminance)
0x80 -2.0 (inverse luminance) .

Return True for success.

MPEGIO2_API bool **MPEGIO2_setSaturation**(unsigned long chanNum, unsigned char saturation);

Function: Set the Preview and MPEG Encoding Video Chrominance saturation on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **saturation**: the new saturation value, within 0 ~ 255, default is 0x40(64):

0x7F 1.984 (maximum) -
0x40 1.0 (ITU level) -
0x00 0 (color off) nominal
0xC0 -1.0 (inverse chrominance) -
0x80 -2.0 (inverse chrominance)

Return True for success.

Note: Saturation set by **MPEGIO2_setSaturation** affetcs overall video on screen and on encoded MPEG stream, but does not affect the Saturation on video loopback output ports Video Loopback RCA Out / Video Loopback S-Video Out on the **Breakout Box**, while **MPEGIO2_setInputUsa/MPEGIO2_setInputVsa** do affect video Loopback RCA/ S-Video Out, although only within one single input source.

MPEGIO2_API bool **MPEGIO2_getSaturation**(unsigned long chanNum, unsigned char &saturation);

Function: Get the Preview and MPEG Encoding Video Chrominance saturation on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **saturation**: the current saturation value, within 0 ~ 255.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setHue**(unsigned long chanNum, unsigned char hue);

Function: Set the Preview and MPEG Encoding Video Colour Hue(Phase) on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **hue**: the new hue value, within 0 ~ 0x80, default is 0x00:

0x7F	….	+178.6
0x00	....	0, nominal
0x80	....	-180.0

Return True for success.

Note: Hue set by **MPEGIO2_setHue** affetcs overall video on screen and on encoded MPEG stream, but does not affect the Hue on video loopback output ports Video Loopback RCA Out and Video Loopback S-Video Out (as labelled on the **Breakout Box**), while **MPEGIO2_setInputHue** does affect video Loopback RCA/S-Video Out, although only within one single input source.

MPEGIO2_API bool **MPEGIO2_getHue**(unsigned long chanNum, unsigned char &hue);

Function: Cet the Preview and MPEG Encoding Video Colour Hue(Phase) on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **hue**: the current hue value, within 0 ~ 0x80, default is 0x00:

0x7F	….	+178.6
0x00	....	0, nominal
0x80	....	-180.0

Return True for success.

MPEGIO2_API bool **MPEGIO2_setSharp**(unsigned long chanNum, unsigned char sharp);
Function: Set the Preview and MPEG Encoding Video colour luminance filter for Sharpness(peaking) adjustment on an **MPEGIO2** Channel.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **sharp**: the new sharpness value, within 0x00 ~ 0x0F.
Return True for success.
Note: Sharpnessue set by **MPEGIO2_setSharp** affetcs overall video on screen and on encoded MPEG stream, but does not Affect the sharpness on video loopback output ports Video Loopback RCA Out and Video Loopback S-Video Out (as labelled on the **Breakout Box**), while **MPEGIO2_setInputSharp** does affect video Loopback RCA/S-Video Out, although only within one single input source.

MPEGIO2_API bool **MPEGIO2_getSharp**(unsigned long chanNum, unsigned char &sharp);
Function: Get the Preview and MPEG Encoding Video colour luminance filter for Sharpness(peaking) adjustment on an **MPEGIO2** Channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **sharp**: the current sharpness value, within 0x00 ~ 0x0F.
Return True for success.

MPEGIO2_API bool **MPEGIO2_setAutoBkColour**(unsigned long chanNum,
bool inputSrc5,
bool automatic);
Function: Set background colour mode on an **MPEGIO2** channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc5**: True to set video input source number 5's background colour mode, false to set video input sources number 1 to 4's background colour mode.
Input Parameter **automatic**: true to set the background colour mode to automatic, false(default) to set to manual
Return True for success.
Note: When automatic mode is enabled, the background colour automatically appears in a video sub window when no signal is from its assigned video input source, otherwise a Video Sub Window has no background colour appearing when no video signal is available in its assigned video input source, until it is manually set by calling function **MPEGIO2_enableBkColour.**

MPEGIO2_API bool **MPEGIO2_getAutoBkColour**(unsigned long chanNum,
bool inputSrc5,
bool &automatic);
Function: Get background colour mode on an **MPEGIO2** channel.

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc5**: True to get video input source number 5's background colour mode, false to get video input sources number 1 to 4's background colour mode.
Output Parameter **automatic**: true means the background colour mode is automatic, false(default) means it is manual.
Return True for success.

MPEGIO2_API bool **MPEGIO2_enableBkColour**(unsigned long chanNum,
bool inputSrc5,
unsigned short subWinNum,
bool enable);
Function: Enable background colour on a video sub window for an MPEGIO2 channel

Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1

Input Parameter **inputSrc5**: true to enable video input number 5's background colour on this video sub window, false to enable video input source numbers 1 to 4's background colour on this video sub window.
Input Parameter **subWinNum**: Video Sub Window Number, must be within 0 ~ 3
Input Parameter **enable**: true to enable background colour, false to disable it
Return true for success.
Note: When a video sub window's background colour is enabled, the background colour appears in the video sub window when no signal is from its assigned video input source, otherwise a video sub window has no background colour appearing when no video signal is available in its assigned video input source. To enable a video sub window's background colour regardless which video input source is assigned to it, call this function twice with this sub window's number and with parameter **inputSrc5** == true and == false respectively.

MPEGIO2_API bool **MPEGIO2_getEnableBkColour**(unsigned long chanNum,
bool inputSrc5,
unsigned short subWinNum,
bool &enable);
Function: Return the Enable/Disable mode of background colour on a video sub window for an MPEGIO2 channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **inputSrc5**: true to get the enable mode of video input number 5's background colour on this video sub window, false to get the enable mode of video input source numbers 1 to 4's background colour on this video sub window.
Input Parameter **subWinNum**: Video Sub Window Number, must be within 0 ~ 3
Output Parameter **enable**: true to enable background colour, false to disable it
Return true for success.

MPEGIO2_API bool **MPEGIO2_setBkColour**(unsigned long chanNum, unsigned char colour);
Function: Set background colour on a channel: background colour appears on the areas where no sub window covers.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **colour**: 0: Black (default), 1: 40% Gray , 2: 75% Gray, 3: 100% Amplitude 100% Saturation Blue.
Return true for success.
Note: Background colour is effective on the entire video frame unrelated to any Sub Window.

MPEGIO2_API bool **MPEGIO2_getBkColour**(unsigned long chanNum, unsigned char &colour);
Function: Set background colour on an **MPEGIO2** channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **colour**: 0: Black (default), 1: 40% Gray , 2: 75% Gray, 3: 100% Amplitude 100% Saturation Blue.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setSubWinBlankColour**(unsigned long chanNum,
unsigned char colour);
Function: Set the colour for Sub-Windows when they have no signal (are blank) on an **MPEGIO2** channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Input Parameter **colour**: 0: Black, 1: 40% Gray , 2: 75% Gray, 3(default): 100% Amplitude 100% Saturation Blue.
Return true for success
Note: Sub Window Blank colour set here is effective on all 4 video sub windows.

MPEGIO2_API bool **MPEGIO2_getSubWinBlankColour**(unsigned long chanNum,
unsigned char &colour);
Function: Get the colour for Sub-Windows when they have no signal (are blank) on an **MPEGIO2** channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Output Parameter **colour**: 0: Black, 1: 40% Gray , 2: 75% Gray, 3: 100% Amplitude 100% Saturation Blue.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setColourBar**(unsigned long chanNum, bool on, bool pathX = true);
Function: Turn on/off the colour bar for a channel
Input parameter **chanNum**: channel number, must be between 0 ~ totalChans-1
Input parameter **on**: True to turn on colour bar, false to turn it off (default)
Input parameter **pathX**: should always set to true for **MPEGIO2.**
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getColourBar**(unsigned long chanNum, bool pathX = true);
Function: Return true if a channel's colour bar is on
Input parameter **chanNum**: channel number, must be between 0 ~ totalChans-1
Input parameter **pathX**: should always set to true for **MPEGIO2.**

MPEGIO2_API bool **MPEGIO2_setColourKill**(unsigned long chanNum, bool on, bool pathX = true);
Function: Turn on/off the colour kill (always display black and white video) for a channel
Input parameter **chanNum**: channel number, must be between 0 ~ totalChans-1
Input parameter **on**: True to turn on colour kill, false to turn it off (default)
Input parameter **pathX**: should always set to true for **MPEGIO2**
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getColourKill**(unsigned long chanNum, bool pathX = true);
Function: Return true if a channel's colour kill is on
Input parameter **chanNum**: channel number, must be between 0 ~ totalChans-1
Input parameter **pathX**: should always set to true for **MPEGIO2**.

MPEGIO2_API bool **MPEGIO2_setInputGain**(unsigned long chanNum,
unsigned char signalLine,
unsigned char gain);
Function: Set video signal input gain on the OSD and Video Input IC.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **signalLine**: 0=Chroma Signal Line, 1=Luma Signal Line, 2=CVBS Signal Line
Input Parameter **gain**: must be 0 ~ 7:

0: 90.625	% of raw signal
1: 93.75	% of raw signal
2: 96.875	% of raw signal
3: 100	% of raw signal
4: 103.125	% of raw signal
5: 106.25	% of raw signal
6: 109.375	% of raw signal
7: 112.5	% of raw signal (this is the default)

Return: True for success
Note: This video gain affects video preview on PC screen, video loopback output ports and encoded MPEG video.

MPEGIO2_API bool **MPEGIO2_getInputGain**(unsigned long chanNum,
unsigned char signalLine,
unsigned char &gain);
Function: Get video signal input gain on the OSD and Video Input IC.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **signalLine**: 0=Chroma Signal Line, 1=Luma Signal Line, 2=CVBS Signal Line
Output Parameter **gain**: within 0 ~ 7: same as in function "**MPEGIO2_setInputGain**"
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setBlackLevel**(unsigned long chanNum, bool is75IRE);
Function: Set video signal input black level on the OSD and Video Input IC.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **is75IRE**: true to set Black Level to 7.5IRE, false (default) to set black level to the same as the blank level
Note 1: When input is non-Japanese NTSC, should set black level to 7.5IRE(**is75IRE**=True), When input is Japanese NTSC or PAL should set black level to 0 IRE (**is75IRE**=False)
Note 2: Video input black level on OSD and Video Input IC affects video preview on PC screen, video loopback output ports and on the encoded MPEG video.
Note 3: This function has no effect on video input from the 5Th source (VIN4 RCA and VIN4 S-Video).

MPEGIO2_API bool **MPEGIO2_getBlackLevel**(unsigned long chanNum, bool &is75IRE);
Function: Get the video signal input black level on the OSD and Video Input IC.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **is75IRE**: true the Black Level is 7.5IRE, false (default) the black level is the same as the blank level.

MPEGIO2_API bool **MPEGIO2_setWhitePeakDetecionThreshold**(unsigned long chanNum,
unsigned char th);

Function: Set the White Peak Detection Threashold for a channel's VIN0~VIN3 Input Source.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Input Parameter **th**: 0 ~ 0xFE: The actual value for White Peak Detection Threashold, Default is 0xD8(216).
0xFF: Use Auomatic White Peak Detection, ignoring all manually set values.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getWhitePeakDetecionThreshold**(unsigned long chanNum,
unsigned char &th);

Function: Get the White Peak Detection Threashold value for a channel's VIN0~VIN3 Input Source.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Output Parameter **th**: 0 ~ 0xFE: The actual value for White Peak Detection Threashold, Default is 0xD8(216).
0xFF: Use Auomatic White Peak Detection, ignoring all manually set values.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setNormalMaximumGain**(unsigned long chanNum,
unsigned char NMGAIN);

Function: Set the Normal AGC loop Maximum Gain
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **NMGAIN**: 0 ~ 0x0F: The Normal AGC loop Maximum Gain, Default is 3.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getNormalMaximumGain**(unsigned long chanNum,
unsigned char &NMGAIN);

Function: Get the current Normal AGC loop Maximum Gain value.
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output Parameter **NMGAIN**: 0 ~ 0x0F: The current Normal AGC loop Maximum Gain.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setWhitePeakGain**(unsigned long chanNum,
unsigned char WPGAIN);

Function: Set the White Peak AGC loop Gain.
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **WPGAIN**: 0 ~ 7: The White Peak AGC loop Gain, Default is 1.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_getWhitePeakGain**(unsigned long chanNum,
unsigned char &WPGAIN);

Function: Set the current White Peak AGC loop Gain.
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output Parameter **WPGAIN**: 0 ~ 7, The current White Peak AGC loop Gain.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_disable5ThInputWhitePeakProtecion**(unsigned long chanNum,
bool disable);

Function: Set the White Peak Protection for a channel's 5th Input Source.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Input Parameter **disable**: true to disable, false to enable(default) the 5th input IC's White Peak Protection.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_get5ThInputWhitePeakProtecion**(unsigned long chanNum,
bool &disable);

Function: Get the White Peak Protection disable/enable status for a channel's 5th Input Source.
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Output Parameter **disable**: true to disable, false to enable(default) the 5th input IC's White Peak Protection.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_disable5ThInputLumaPeakProtecion**(unsigned long chanNum, bool disable);

Function: Disbale or Enable the 5th Video Input IC Luminance Peak Protection.
Input parameter **chanNum**: 0 ~ totalChans-1: the chanel number.
Input parameter **disable**: true to disable the 5^{th} Video Input Device IC's Luma Peak Protection, false to enable it (Default).
Return: True for success.

MPEGIO2_API bool **MPEGIO2_get5ThInputLumaPeakProtecion**(unsigned long chanNum, bool &disable);

Function: Get the Disbale or Enable status of the 5th Video Input IC Luminance Peak Protection.
Input parameter **chanNum**: 0 ~ totalChans-1: the chanel number.
Output parameter **disable**: true to disable the 5^{th} Video Input Device IC's Luma Peak Protection, false to enable it (Default).
Return: True for success.

MPEGIO2_API bool **MPEGIO2_set5ThInputLumaAGC**(unsigned long chanNum, unsigned char lumaAGC);

Function: Set the Luminance (Luma) AGC (Automatic Gain Control) for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **lumaAGC**: 0 = Luma AGC Disabled (fixed gain value)
1 = Luma AGC enabled (default)
2 = Luma AGC frozen to the previously set value
Return true for success.

MPEGIO2_API bool **MPEGIO2_get5ThInputLumaAGC**(unsigned long chanNum, unsigned char &lumaAGC);

Function: Get the Luminance (Luma) AGC (Automatic Gain Control) setup for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **lumaAGC**: Same as in **MPEGIO2_set5ThInputLumaAGC**
Return true for success.

MPEGIO2_API bool **MPEGIO2_set5ThInputChromaAGC**(unsigned long chanNum, unsigned char chromaAGC);

Function: Set the Colour (Chroma) AGC (Automatic Gain Control) for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **chromaAGC**: 0 = Chroma AGC Disabled (fixed gain value)
1 = Chroma AGC enabled (default)
2 = Chroma AGC frozen to the previously set value
Return true for success.

MPEGIO2_API bool **MPEGIO2_get5ThInputChromaAGC**(unsigned long chanNum, unsigned char &chromaAGC);

Function: Get the Colour (Chroma) AGC (Automatic Gain Control) setup for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **chromaAGC**: Same as in **MPEGIO2_set5ThInputChromaAGC**
Return true for success.

MPEGIO2_API bool **MPEGIO2_set5ThInputROMPatchCode**(unsigned long chanNum);

Function: Set the ROM Patch Code (program patch running inside the IC) for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number.
Return true for success
Note 1: This Function Only works for PCB ver num >= 2.
Note 2: This function automatically set the Use Latest Patch Code Status to True(calling function **MPEGIO2_5ThInputROMPatchCodeUse** with parameter Enable==True) for the 5th Video Input IC
Note 3: To delete the 5Th Video Input IC's loaded latest ROM Code and revert back to the original ROM Patch Code (version 0): call **MPEGIO2_5ThInputROMPatchCodeUse** with its parameter enable==false then reboot the PC.

MPEGIO2_API bool **MPEGIO2_get5ThInputROMVer**(unsigned long chanNum, byte &ROMVer);

Function: Get the ROM Version for the 5th Video Input IC
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **ROMVer**: 0 = unpatched (original ROM), non-zero are patched(downloaded) ROM (latest ver. is 0x8C).
Return true for success.

MPEGIO2_API bool **MPEGIO2_5ThInputROMPatchCodeUse**(unsigned long chanNum,
bool enable);

Function: Enable/Disable to use the Latest ROM Update Code for the 5Th Input Device Decoder IC
(This Function Only works for ver num >= 2 PCB Schematics)

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Input parameter **enable**: True to use the latest (Version 0x8C) Patch Code, False(Default): Not to use this Patch Code (keep the ROM Version 0)

Return true for success.

Note 1: This Function can be called any time, regardless function **MPEGIO2_initSDK**() is called or not

Note 2: This function simply set the Enable/Disable Status but does not actually Set the ROM Patch Code(download the firmware to the IC) itself for the 5Th Video Input IC

Note 3: When using the Patch Code is Enabled, each time the SDK is started or the 5Th Input IC is Reset the Latest ROM Patch Code is downloaded to this IC, otherwise no Patch Code will be downloaded to this IC

Note 4: Some models of the 5Th Input IC could have bad video signal when using or not using the Latest ROM Patch Code, when that happens, call this function to disable or enable the Patch Code then Reboot the PC (See Note 5) could rectify the bad signal problem

Note 5: To disable using the Latest ROM Patch Code on the 5Th Video Input IC: Call this function with enable==False then reboot the PC: simply reset the IC will Not revert its ROM Code back to the original version 0!

Note 6: Calling function **MPEGIO2_set5ThInputROMPatchCode** will automatically set the "Enable" status for setting the Latest ROM Patch Code to True (as well as actually set the latest ROM Patch Code).

MPEGIO2_API bool **MPEGIO2_5ThInputROMPatchCodeUseStatus**(unsigned long chanNum,
bool &enable);

Function: Return the Enable/Disable Status for using the Latest ROM Update Code for the 5Th Input Device Decoder IC
(This Function Only works for ver num >= 2 PCB Schematics)

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Output parameter **enable**: Receive True: use the latest (Version 0x8C) Patch Code,
Receive False(Default): Not to use this Patch Code (keep the ROM Version 0)

Return true for success.

Note: This Function can be called any time, regardless function **MPEGIO2_initSDK**() is called or not.

MPEGIO2_API bool **MPEGIO2_setVideoAGCGain**(unsigned long chanNum,
unsigned short inputSrc,
bool AGCEnable,
unsigned short AGCGain);

Function: Enable/Disable Auto Video AGC(Automatic Gain Controller) Gain on an Input Source VIN0 ~ VIN3,
when Video AGC Gain is disabled, also set the manual AGC Gain value to **AGCGain**.

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Input Parameter **inputSrc**: 0 ~ 3 for Input Source VIN0 ~ VIN3

Input Parameter **AGCEnable**: True(Default) to enable Automatics AGC Gain for Video on the Input Source,
false to disable Auto AGC Gain (manually use the **AGCGain** value to set the gain).

Input Parameter **AGCGain**: Only meaningful when **AGCEnable** is False:
set the AGC gain value used for the Input Source's Video Signal: valid values are 0 ~ 511.

Return true for success.

MPEGIO2_API bool **MPEGIO2_getVideoAGCGain**(unsigned long chanNum,
unsigned short inputSrc,
bool &AGCEnable,
unsigned short &AGCGain);

Function: Get the Enable/Disable Video AGC(Automatic Gain Controller) status on Input Source VIN0 ~ VIN3,
when Video Auto AGC Gain is disabled, also get the manual AGC Gain value in AGCGain.

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Input Parameter **inputSrc**: 0 ~ 3 for Input Source VIN0 ~ VIN3

Output Parameter **AGCEnable**: get True(Default) for enabling AGC Gain for Video on the Input Source, false to disable it.

Output Parameter **AGCGain**: Get AGC gain value used for the Input Source's Video Signal (if Auto AGC Gain is disabled).

Return true for success.

4.3.5 Video Size & Position Functions

MPEGIO2_API bool **MPEGIO2_setVideoSrcCrop**(unsigned long chanNum,
int src,
unsigned short left,
unsigned short top,
unsigned short width,
unsigned short height);

Function: Set the video cropping size for a video input source on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **src**: 0~4 for input1~input5 for one of the the 5 Video Input Sources, if **src** is 5 it's treated as 4.
Input parameter **left**: the left position of the cropped video size in pixel unit, default is 32
Input parameter **top** : the top position of the cropped video size in pixel unit, default is 5 for PAL and 6 for NTSC
Input parameter **width**: the horizontal size of the cropped video size in pixel unit, default is 720
Input parameter **height**: the vertical size of the cropped video size in pixel unit, default is 576 for PAL and 480 for NTSC
Return true if succeed
Note 1: left + width must be < 864 for PAL, must be < 858 for NTSC,
top + height must be < 624 for PAl, must be < 524 for NTSC
Note 2: When **src**=4, see **MPEGIO2_set5ThInputAutoCrop.**

MPEGIO2_API bool **MPEGIO2_getVideoSrcCrop**(unsigned long chanNum,
int src,
unsigned short &left,
unsigned short &top,
unsigned short &width,
unsigned short &height);

Function: Get the video cropping size for a video input source on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **src**: 0~4 for input1~input5, if **src** is 5 it's treated as 4.
Output parameter **left**: the left position of the cropped video size in pixel unit, default is 32
Output parameter **top** : the top position of the cropped video size in pixel unit, default is 5 for PAL and 6 for NTSC
Output parameter **width**: the horizontal size of the cropped video size in pixel unit, default is 720
Output parameter **height**: the vertical size of the cropped video size in pixel unit, default is 576 for PAL and 480 for NTSC
Return true if succeed.

MPEGIO2_API bool **MPEGIO2_setHVideoStartPos**(unsigned long chanNum, unsigned long pos);
Function: Set the Video Horizontal Start Position on an **MPEGIO2** Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pos**: the new Video Horizontal Start Position value in pixels, within 0 ~ 719.
Return true for success
Note: This setting will affect the left video starting position of video preview, video encoding and still image capture.

MPEGIO2_API bool **MPEGIO2_setVVideoStartPos**(unsigned long chanNum, unsigned long pos);
Function: Set the Video Vertical Start Position on an **MPEGIO2** Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pos**: the new Video Vertical Start Position value in pixels, within 0 ~ 719.
Return true for success
Note: The top video starting position of video preview, video encoding and still image capturing
will all be affected by this setting.

MPEGIO2_API bool **MPEGIO2_getVideoStartPos**(unsigned long chanNum,
unsigned long &HPos,
unsigned long &VPos);

Function: Get the Video Horizontal and Vertical Start Positions on a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **HPos**: the current Video Horizontal Start Position value in pixels.
Output parameter **VPos**: the current Video Vertical Start Position value in pixels.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setVideoZooming( unsigned long chanNum,
                                           bool enable,
                                           bool horizontalOnly,
                                           bool zoomBoundary,
                                           unsigned char boundColour,
                                           bool zoomMark,
                                           unsigned char zoomMarkEffect,
                                           unsigned short x = 0,
                                           unsigned short y = 0);
```

Function: Enable / disable video zooming within the video frame for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **enable**: true to enable 2-times video zooming, false to disable zooming.
Input parameter **horizontalOnly**: true to zoom at horizontal direction only, false to zoom both horizontally and vertically
Input parameter **zoomBoundary**: true to enable a boundary around the zoomed area, false disable the boundary
Input parameter **boundColour**: when zoomBoundary is true, this value defines the colour of the boundary:
0=0% Black, 1=25% Gray, 2=75% Gray, 3=100% White
Input parameter **zoomMark**: true to enable zoom mark(see **zoomMarkEffect**), false to disable zoom mark
Input parameter **zoomMarkEffect**: Set the marking effect(if parameter **zoomMark** is true) for zoomed area:
0=10 IRE bright up for inside of zoomed area (default)
1=20 IRE bright up for inside of zoomed area
2=10 IRE bright up for outside of zoomed area
3=20 IRE bright up for outside of zoomed area
Input parameter **x / y**: the zomming start point within the video frame:
x must be within 0 ~ 715, y must be within 0 ~ 575 for PAL, within 0 ~ 475 for NTSC

Zooming points are in 4-pixel unit: i.e., x=0 or 1 or 2 or 3 have the same result,
x=4/5/6/7 have the same result,
similarly, y=0 or 1 or 2 or 3 have the same result, y=4/5/6/7 have the same result.

Return true if succeed
Note: The zoomed area is from point (x, y) towards the lower-right edge: (719, 575) for PAL or (719, 479) for NTSC.

```
MPEGIO2_API bool  MPEGIO2_getVideoZooming(unsigned long chanNum,
                                          bool &enable,
                                          bool &horizontalOnly,
                                          bool &zoomBoundary,
                                          unsigned char &boundColour,
                                          bool &zoomMark,
                                          unsigned char &zoomMarkEffect,
                                          unsigned short &x,
                                          unsigned short &y);
```

Function: Get video zooming parameters for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **enable**: true to enable 2-times video zooming, false to disable zooming.
Output parameter **horizontalOnly**: true to zoom at horizontal direction only, false to zoom both horizontally and vertically
Output parameter **zoomBoundary**: true to enable a boundary around the zoomed area, false disable the boundary
Output parameter **boundColour**: when zoomBoundary is true, this value defines the colour of the boundary:
0=0% Black, 1=25% Gray, 2=75% Gray, 3=100% White
Output parameter **zoomMark**: true to enable zoom mark(see zoomMarkEffect), false to disable zoom mark
Output parameter **zoomMarkEffect**: Set the marking effect(if parameter **zoomMark** is true) for zoomed area:
0=10 IRE bright up for inside of zoomed area (default)
1=20 IRE bright up for inside of zoomed area
2=10 IRE bright up for outside of zoomed area
3=20 IRE bright up for outside of zoomed area
Output parameters **x / y**: the zomming start point within the video frame.
Return true if succeed.

MPEGIO2_API bool **MPEGIO2_set5ThInputVBS**(unsigned long chanNum,
char vbstart,
char vbstop);

Function: Set The 5Th Video Input Source's Vertical blanking (VBLK) start/stop values that will affect the 5Th Input Source's Video vertical start position and height:
these are adjustable with respect to the standard vertical blanking intervals.

Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input Parameter **vbstart**: as a signed char value: -128 ~ +127 == The Vertical Blank Area will appear (in number of lines of displayed video) before (when value is negative) or after (when value is positive) the standard vertical blanking intervals.
Default is 0: Same time as start of the standard vertical blanking interval

Input Parameter **vbstop**: as a signed char value: -128 ~ +127 == The Vertical Blank Area will end (in number of lines of displayed video) before (when value is negative) or after (when value is positive) the standard vertical blanking intervals.
Default is 0: Same time as stop of the standard vertical blanking interval

Return: True for success.

Note: This setting affects both the RCA and SVideo Input on the 5Th Video Source (different external video incoming sources should not connect to the 5Th Video Input's RCA and SVideo sockets simultaneously) .

MPEGIO2_API bool **MPEGIO2_get5ThInputVBS**(unsigned long chanNum,
char &vbstart,
char &vbstop);

Function: Retrieve the The 5Th Video Input Source's Vertical blanking (VBLK) start/stop values

Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Output Parameter **vbstart**: the retrived VBStart value

Output Parameter **vbstop**: the retrived VBStop value

Return: True for success.

MPEGIO2_API bool **MPEGIO2_set5ThInputHPos**(unsigned long chanNum, int hstart, int hstop);

Function: Set The 5Th Video Input Source's Horizontal start/stop values that will affect the 5Th Input Source's Video pixel horizontal start position and width.

Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input Parameter **hstart**: as a signed integer value: -512 ~ +511 == the number of pixels to blank out before/after the first horizontally active pixel. Default is 0.

Input Parameter **hstop**: as a signed integer value: -512 ~ +511 == the number of pixels to blank out before/after the last horizontally active pixel. Default is 0.

Return: True for success.

MPEGIO2_API bool **MPEGIO2_get5ThInputHPos**(unsigned long chanNum, int &hstart, int &hstop);

Function: Get The 5Th Video Input Source's Horizontal start/stop values that will affect the 5Th Input Source's Video pixel horizontal start position and width.

Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number

Output Parameter **hstart**: a signed integer value: -512 ~ +511 == the number of pixels to blank out before/after the first horizontally active pixel

Output Parameter **hstop**: a signed integer value: -512 ~ +511 == the number of pixels to blank out before/after the last horizontally active pixel

Return: True for success.

MPEGIO2_API bool **MPEGIO2_set5ThInputAutoCrop**(unsigned long chanNum,
bool autoCrop,
BYTE VSize);

Function: Set The 5Th Video Input automatic cropping on or off

Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input Parameter **autoCrop**: true(default) to enable, false to disable automatic cropping

Input Parameter **VSize**: Only meaningful when **autoCrop** is true: Select the horizontal active video size for the 5Th Input:
0=720 Pixels(default), 1=704 Pixels, 2 or 3=640 Pixels

Return: True for success

Note: When **MPEGIO2_setVideoSrcCrop** is called with parameter **src**==4,
the 5Th Video Input's automatic cropping is disabled.

MPEGIO2_API bool **MPEGIO2_get5ThInputAutoCrop**(unsigned long chanNum,
bool &autoCrop,
BYTE &VSize);

Function: Get The 5Th Video Input automatic cropping on or off mode
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **autoCrop**: true to enable, false to disable autimatic cropping
Output Parameter **VSize**: Only meaningful when **autoCrop** is true: Receive the horizontal active video size for the 5Th Input:
0=720 Pixels, 1=704 Pixels, 2 or 3=640 Pixels
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setVideoVertDelay**(unsigned long chanNum, unsigned char delay);
Function: Set vertical line delay of the active input video on the OSD and Video Input IC for a channel
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **delay**: must be within 0 ~ 31:
0 = -11 Lines delayed
: :
12 = 0 Line delayed (default)
: :
31 = +19 Lines delayed.
Return: True for success.
Note: this setting will shift the input video frame up/down vertically.

MPEGIO2_API bool **MPEGIO2_getVideoVertDelay**(unsigned long chanNum, unsigned char &delay);
Function: Get vertical line delay of the active input video on the OSD and Video Input IC for a channel
Input Parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output Parameter **delay**: can be within 0 ~ 31: same as in function "**MPEGIO2_setVideoVertDelay**".
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setVideoHoriDelay**(unsigned long chanNum, unsigned char delay);
Function: Set horizontal pixel delay of the active input video on the OSD and Video Input IC for a channel
Input Parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **delay**: must be within 0 ~ 63:
0 = - 32 Pixel delay
: :
32= 0 Pixel delay (default)
: :
63= + 31 Pixel delay
Return: True for success.
Note: this setting will shift the input video frame left/right horizontally.

MPEGIO2_API bool **MPEGIO2_getVideoHoriDelay**(unsigned long chanNum, unsigned char &delay);
Function: Get horizontal pixel delay of the active input video on the OSD and Video Input IC for a channel
Input Parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output Parameter **delay**: within 0 ~ 63: same as in function "**MPEGIO2_setVideoHoriDelay**".
Return: True for success.

MPEGIO2_API unsigned long **MPEGIO2_getVideoFrameHeight**(unsigned long chanNum);
Function: Return the video frame height according to the signal type currently used by the Video Preview IC
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return: 576 for PAL signal, 480 for NTSC signal
Note: by default the reurned value is 576, inc. failure of this function (wrong chanNum etc.)

4.3.6 Video Output Port Functions

Note 1: The SDK doesn't automatically save the settings made by the following functions and it always uses default values for them: the application software need to remember any settings it made in these functions if it wishes to keep them.

Note 2: In the following functions, the 1st **Output IC** is the MPEG Decode RCA/SVideo Output Ports **J11/J6/J3/J8** on the Breakout Box, the 2nd **Output IC** is the Loopback RCA/SVideo Output Ports **J9/J1/J5/J17** on the Breakout Box.

MPEGIO2_API bool **MPEGIO2_setOutputICColourBar**(unsigned long chanNum,
bool OutIC1,
bool colourBarOn);

Function: Set the 1st or 2nd Output IC to display Colour Bar.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **OutIC1**: True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **colourBarOn**: True to turn on, false to turn off the Colour Bar on the IC's Output Ports(RCA and SVideo).
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setOutputICCHPS**(unsigned long chanNum,
bool OutIC1,
unsigned char CHPS);

Function: Set the 1st or 2nd Output IC's phase of encoded colour subcarrier including burst.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **OutIC1** True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **CHPS**: phase of encoded colour subcarrier relative to horizontal sync; default 0 for PAL, 0x77 for NTSC.
Return: True for success.

MPEGIO2_API bool **MPEGIO2_setOutputICBlankLevel**(unsigned long chanNum,
bool OutIC1,
unsigned char BNLV);

Function: Set the 1st or 2nd Output IC's Blanking Level
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **OutIC1**: True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **BNLV**: within 0 ~ 0x3F , Output Blanking Level, Default is 0x2A: smaller value seems to reduce flickering.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOutputICBlackLevel**(unsigned long chanNum,
bool OutIC1,
unsigned char BALV);

Function: Set 1st or 2nd Output IC's Black Level
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **OutIC1**: True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **BALV**: within 0 ~ 0x3F , Output Black Level, Default is 0x2A for NTSC, 0x23 forPAL.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOutputICUGain**(unsigned long chanNum,
bool OutIC1,
unsigned char gain);

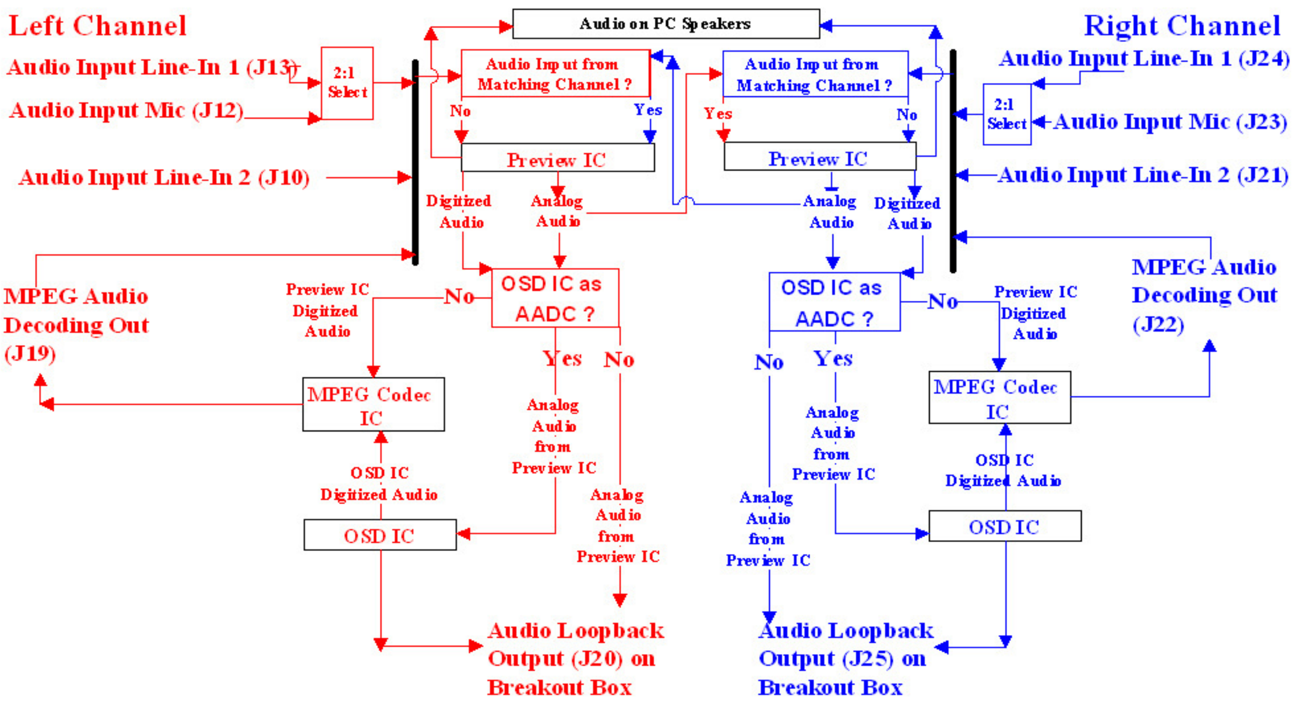
Function: Set the 1st or 2nd Output IC's Video U Component (Cb Signal) Gain.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **OutIC1**: True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **gain**: U component gain within 0 ~ 255, Default is 0x76 for NTSC, 0x7D forPAL,
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOutputICVGain**(unsigned long chanNum,
bool OutIC1,
unsigned char gain);

Function: Set the 1st or 2nd Output IC's Video V Component (Cr Signal) Gain. Input
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **OutIC1**: True to set the 1st Output IC, false to set the 2nd Output IC.
Input parameter **gain**: V component gain within 0 ~ 255, Default is 0xA5 for NTSC, 0xAF forPAL,
Return true for success.

4.4 Audio I/O Functions

Every **MPEGIO2** channel has 4 possible Audio Input Sources: the 1st audio input source is selectable between "**Audio 1st Input Mic-In**" and "**Audio 1st Input Line-In**" sockets; the 2nd audio input source is the **Audio 2nd Line-In** socket; the 3rd audio input source is the MPEG decoding audio output, and the 4th input source is the audio output from this channel's matching channel on the same **MPEGIO2** card. Audio signals from these input sources can be mixed to be simultaneously heard on PC's speakers and the audio Loopback Output port on the Breakout Box, as well as to be encoded into MPEG data streams, as illustrated below in the "**Audio I/O Diagram**":



MPEGIO2 Audio I/O Diagram

MPEGIO2_API bool **MPEGIO2_initAudioChips**(unsigned long chanNum);
Function: Initialize audio input hardware for one **MPEGIO2** channel.
Input Parameter **chanNum**: **MPEGIO2** channel num, 0 ~ totalChans - 1
Return true for success.
Note: This function must be called at least once for an **MPEGIO2** channel before all other audio input functions become effective and the audio input signals can be monitored, previewed and encoded.

MPEGIO2_API bool **MPEGIO2_setAudioStandard**(unsigned long chanNum,
unsigned long standard,
unsigned long samplingRate);
Function: Set input audio standard to a channel for preview (not affecting MPEG audio encoding)
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **standard**: 1 for NTSC, 2 for PAL, default audio standard is PAL
Input parameter **samplingRate**: 0 or 1=32KHz(default), 2=44.1KHz, 3=48KHz, sampling rate for Preview/Monitoring.
Return: true for success.

MPEGIO2_API bool **MPEGIO2_getAudioSamplingRate**(unsigned long cardNum,
unsigned long &samplingRate);
Function: Get input audio sampling rate currently used for audio preview on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **samplingRate**: lowest 2 bits indicate audio preview sampling rate: 0 or 1=32Khz, 2=44.1KHz, 3=48KHz
Return: true for success.

MPEGIO2_API bool **MPEGIO2_AudioLineInMic**(ULONG chanNum, bool lineIn);
Function: Select Either Line-In or Mic-In as the "1st Audio Input Source": Note Microphone is Not available currently!
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Input Parameter **lineIn**: If true, select **Audio 1st Input Line-In** as the "1st Audio Input Source":
the J13(chanNum is LeftChan)/J24(chanNum is RightChan) socket on the **Breakout Box**,
If false(default), select **Audio 1st Input Mic-In** (Microphone Input) as the "1st Audio Input Source".
Return: true for success.
Note: "1st Audio Input Source" is the 1st of the 4 possible input sources that can be used as a channel's audio input,
the other 3 input sources are:
2nd Audio Input Source -- **Audio 2nd Line-In** socket on the **Breakout Box:**
the J10(chanNum is LeftChan)/J21(chanNum is RightChan) socket on the **Breakout Box**;
3rd Audio Input Source -- MPEG Decoding Audio Output;
4th Audio Input Source -- Audio Output from the Matching Channel on the same **MPEGIO2** card;
See function **MPEGIO2_AudioInputSelect**() on how to select these 4 sources as audio input .

MPEGIO2_API bool **MPEGIO2_AudioLineInMicStatus**(ULONG chanNum, bool &lineIn);
Function: Get the Line-In or Mic-In status of the "1st Audio Input Source"
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Output Parameter **lineIn**: True means the **Audio 1st Input Line-In** socket is "1st Audio Input Source".
False means the **Audio 1st Input Mic-In** is the "1st Audio Input Source".
Return: true for success.

MPEGIO2_API bool **MPEGIO2_AudioInputSelect**(ULONG chanNum, unsigned char src);
Function: Select Audio Input Source for an **MPEGIO2** Channel
Input Parameter **chanNum**: the channel number, must be between 0 ~ totalChans-1
Input Parameter **src**: Setting the lowest 4 bits of this parameter to indicate which input source will be used as audio input:
bit 0 is the "**Audio 1st Input Source**" as set by function **MPEGIO2_AudioLineInMic;**
bit 1 is the **Audio 2nd Line-In** socket: the J10/J21 socket on the **Breakout Box**;
bit 2 is the MPEG Decoding Audio Output when this channel is doing MPEG decoding;
bit 3 is the Audio ouput from the matching channel on same **MPEGIO2** card.

More than one of these bits can be selected to simultaneously mix multiple audio input sources:
e.g. set **src** == 3 will mix "**Audio 1st Input Source**" and "**Audio 2nd Line-In**" as input.
If all the lowest 4 bits of **src** are zeroes then this function does not select any input and returns false.
Return: true for success.
Note: When the channel is not doing MPEG decoding, the MPEG Decoding Audio Output could contain random noise.

MPEGIO2_API bool **MPEGIO2_setAudioInLine**(unsigned long chanNum, unsigned long lineNum);
Function: Select Audio Input Source for an **MPEGIO2** Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **lineNum**: 0 (default)= the normal audio input from this channel's input sockets as selected by
calling function **MPEGIO2_AudioInputSelect.**
1 = the matching channel(on the same **MPEGIO2** card)'s audio output.
Note if chanNum is the left channel, "the macthing channel" will be the right channel ,
if chanNum is the right channel, "the macthing channel" will be the left channel.
Return true for success.
Note 1 : When this function set a channel's Audio Input Line to 0 (this is the default),
the actual input audio is from socket selected by function **MPEGIO2_AudioInputSelect**
Note 2**:** To successfully set a channel to use its matching channel's audio output as its own audio input, not only this
function need to be called with parameter **lineNum** as 1, function **MPEGIO2_AudioPreviewOutSelect** should also
be called using the the matching channel's channel number and parameter "**toSpeakers**" set as false.
Note 3: The "**2nd audio input**" (Not to be confused with the "**Audio 2nd Line-In** socket" on the Breakout Box) of an
MPEGIO2 channel is an internal socket that can be selected by calling this function and function
MPEGIO2_AudioPreviewOutSelect together, it is not visible externally on the **Breakout Box**.
Note 4: Calling this function will also automatically set "audio monitor source" accordingly (see function
"**MPEGIO2_setAudioMonitorOutSrc**") for the "Audio Loopback Output" socket of this channel:
if audio Input Line is set to the normal audio input from this channel's input sockets (lineNum==0),
MPEGIO2_setAudioMonitorOutSrc will be automatically called with parameter **src**=0;
if audio Input Line is set to the matching channel's audio output on the same **MPEGIO2** card(lineNum==1),
MPEGIO2_setAudioMonitorOutSrc will be automatically called with parameter **src**=1.
Note other function could later change this channel's Audio Monitor Output Signal Source thus negating this result.

Note 5: When OSD IC is Audio ADC(see **MPEGIO2_OSDICasAudioADC**), **MPEGIO2_setAudioMonitorOutSrc** must be called with different values for parameter **src** to make audio to be (or not to be) encoded into MPEG data. See function **MPEGIO2_setAudioMonitorOutSrc** for more on this.

MPEGIO2_API bool **MPEGIO2_getAudioInLine**(unsigned long chanNum,
unsigned long &lineNum);

Function: Retrieve the Audio Input Source value of an **MPEGIO2** Channel
Input Parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **lineNum**: 0 = normal audio input from this channel's input sockets.
1 = the matching channel(on the same **MPEGIO2** card)'s audio output.
Return true for success.

MPEGIO2_API bool **MPEGIO2_AudioPreviewOutSelect**(ULONG chanNum, bool toSpeakers);

Function: Select audio preview output of a channel is connected to **Audio Loopback Output** socket on the **Breakout Box,** or connected to the "**2nd audio input**" of the matching channel on the same **MPEGIO2** card
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Input Parameter **toSpeakers**: true(default) to connect audio preview output to **Audio Loopback Output** socket, false to connect audio preview output to the "**2nd audio input**" of the matching channel on the same PCIe card.
Return: true for success
Note 1: To enable a channel to accept its matching channle's audio preview output as its audio input, this function must be called using the the matching channel's channel number and parameter "**toSpeakers**" set as false, then call function **MPEGIO2_setAudioInLine**
Note 2: The "**2nd audio input**" of an **MPEGIO2** channel is an internal socket which can be selected by calling this function and function **MPEGIO2_setAudioInLine** together, it is not visible externally on the **Breakout Box**.
Note 3: When **toSpeakers** is set to false, the "Audio Loopback Output" J20(If **chanNum** is Left Channel on the PCIe card) or J25(If **chanNum** is Right Channel on the PCIe card) socket on the Breakout Box will have no audio output.

MPEGIO2_API bool **MPEGIO2_AudioPreviewOutToSpeakers**(ULONG chanNum,
bool &toSpeakers);

Function: Return if this channel's audio preview output is connected to **Audio Loopback Output** socket, or is connected to the "**2nd audio input**" of the matching channel on the same MPEGIO2 card
Input Parameter **chanNum**: the MPEGIO2 channel number, must be between 0 ~ totalChans-1
Output Parameter **toSpeakers**: true = this channel's audio preview output is connecting to **Audio Loopback Output** socket,
false = this channel's audio preview output is connecting to the "**2nd audio input**" of the matching channel on the same **MPEGIO2** card
Return: true for success.

MPEGIO2_API bool **MPEGIO2_setAudioMonitorOutSrc**(unsigned long chanNum,
unsigned char src);

Function: Set the "**Audio Loopback Output**" socket's signal source
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **src**: 0: the audio signal source is from this channel's audio input sockts, this is the default setting
1: the audio signal source is from the matching channel's audio output
2: no signal is output to "**Audio Loopback Output**" socket

When OSD IC is used as Audio ADC (by calling function **MPEGIO2_OSDICasAudioADC**(chanNum, true)) then this function will also decide if its input audio can be encoded during MPEG encoding process:

(1) If this channel is using its matching channel's audio as its audio input (via calling function **MPEGIO2_setAudioInLine**(chanNum, 1)), this function must be called with parameter **src**=1 to encode audio;
(2) If this channel is using its own audio sockets on the Breakout Box as audio input source, this function must be called with parameter **src**= 0 to make audio to be encoded into MPEG data stream;
(3) If this function is called with parameter **src**= 2 then no audio will be encoded during MPEG encoding.

Return true for success.

MPEGIO2_API bool **MPEGIO2_getAudioMonitorOutSrc**(unsigned long chanNum,
unsigned char &src);

Function: Get the "**Audio Loopback Output**" socket's signal source
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **src**: 0: the signal source is from this channel's audio input sockts
1: the signal source is from the matching channel's audio output
2: no signal is output to "**Audio Loopback Output**" socket

Return true for success.

MPEGIO2_API bool **MPEGIO2_AudioInLeftRightMap**(ULONG chanNum,
unsigned char audioSocket,
unsigned char mapping);

Function: Set up the Left/Right channel mapping of a Stereo Audio Input Socket for an **MPEGIO2** Channel
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Input Parameter **audioSocket**: 0 or 1, indicating which stereo audio input socket to set its left/right stereo channel mapping:
0 for "**Audio 1st Input Line-In**": the J13(chanNum is LeftChan)/J24(chanNum is RightChan) socket on the Breakout Box,
1 for "**Audio 2nd Input Line-In**": the J10(chanNum is LeftChan)/J21(chanNum is RightChan) socket on the Breakout Box.
Input Parameter **mapping**: 4 possible mapping values:

0 : the left channle of the stereo audio input socket connects to the left channel of the **MPEGIO2** audio input, and the right channle of the stereo audio input socket connects to the right channel of the **MPEGIO2** audio input.

1 : the right channle of the stereo audio input socket connects to the left channel of the **MPEGIO2** audio input, and the right channle of the stereo audio input socket connects to the right channel of the **MPEGIO2** audio input. (The left channel of the stereo audio input socket is not used).

2 : the left channle of the stereo audio input socket connects to the left channel of the **MPEGIO2** audio input, and the left channle of the stereo audio input socket connects to the right channel of the **MPEGIO2** audio input. (The right channel of the stereo audio input socket is not used).

3 : the left channle of the stereo audio input socket connects to the right channel of the **MPEGIO2** audio input, and the right channle of the stereo audio input socket connects to the left channel of the **MPEGIO2** audio input.

Return: true for succes.

MPEGIO2_API bool **MPEGIO2_setAudioChanGain**(unsigned long chanNum,
unsigned long audioChanNum,
unsigned char LRB,
unsigned short gain);

Function: Set audio channel's input gains for an **MPEGIO2** channel
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Input Parameter **audioChanNum**: must be values 0 ~ 4 :

Values 0 ~ 2 represent the 3 possible audio input sources (see function **MPEGIO2_AudioInputSelect**):
value 0 is the "1st Audio Input Source" as set by function "**MPEGIO2_AudioLineInMic**";
value 1 is the "**Audio 2nd Input Line-In**" socket on the top Layer of the **Breakout Box**;
value 2 is the MPEG Decoding Audio Output when this channel is doing MPEG decoding;

value 3 is for the MPEG decoding audio output towards external speakers, but seeting this gain also effects MPEG decoding audio heard on PC's speakers via sound card;

value 4 is for audio coming from the audio output of the matching channel on the same **MPEGIO2** card.

Input Parameter **LRB**: The stereo audio's sub-channel on which the volume is to be set:
0 = Left Sub-Channel, 1 = Right Sub-Channel, 2 = Both Left and Right Sub-Channenls
Input Parameter **gain**: audio volume gain, 0(minimum gain - low volume) ~ 79(maximum gain - high volume)
Return true for success.

MPEGIO2_API bool **MPEGIO2_getAudioChanGain**(unsigned long chanNum,
unsigned long audioChanNum,
unsigned short &leftGain,
unsigned short &rightGain);

Function: Get audio channel's volume gains for an **MPEGIO2** channel
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Input Parameter **audioChanNum**: must be values 0 ~ 4 :

0~2 are for the 3 possible audio input sources (see function **MPEGIO2_AudioInputSelect**):
value 0 is the "1st Audio Input Source" as set by function "**MPEGIO2_AudioLineInMic**";
value 1 is the "**Audio Loopback Output**" socket;
value 2 is the MPEG Decoding Audio Output when this channel is doing MPEG decoding;
value 3 is for the MPEG decoding audio output towards external speakers

(for value 3 the gain set will also effect MPEG decoding audio heard on PC's speakers via sound card);

value 4 is for audio coming from the audio output of the matching channel on the same **MPEGIO2** card.
Output Parameter **leftGain**: the stereo audio channel's left sub-channel's volume gain, 0(min) ~ 79(max)
Output Parameter **rightGain**: the stereo audio channel's right sub-channel's volume gain, 0(min) ~ 79(max)
Return true for success.

MPEGIO2_API bool **MPEGIO2_setAudioChanMute**(unsigned long chanNum,
bool inputChans,
bool muteOn);

Function: Mute left and right sub-channels for some stereo audio input or output channels on an **MPEGIO2** channel
Input Parameter **chanNum**: the **MPEGIO2** channel number from 0 ~ totalChans-1
Input Parameter **inputChans**: **=True** for all 1 ~ 3 stereo audio **input** channels(these 3 channels are muted/unmuted together):
1.the "1st Audio Input Source" as set by function "**MPEGIO2_AudioLineInMic**":
This can be either the "Audio 1st Input Line-In" or "Audio 1st Input Mic-In" socket
on the **Breakout Box** Top Layer;
2.the "Audio 2nd Input Line-In" socket on the **Breakout Box** Top Layer;
3.the MPEG Decoding Audio Output fed to Audio Preview Input when this channel is doing MPEG decoding;
=False for 2 audio **output** channels(these 2 channels are muted/unmuted together):
1.the MPEG Decoder stereo audio output channel (from MPEG decoder to external speakers): this means the "MPEG Decode Audio Out" stereo socket on the **Breakout Box** Bottom Layer ;
2.the audio coming from the matching channel's audio out connected into this channel's MPEG decoder stereo audio output (the "MPEG Decode Audio Out" stereo socket).
Input Parameter **muteOn**: true to mute, false to un-mute
Return true for success.

MPEGIO2_API bool **MPEGIO2_getAudioChanMute**(unsigned long chanNum,
bool &inputAudioMuted,
bool &outputChanMuted);

Function: Get the stereo audio channels' mute status on an **MPEGIO2** channel
Input Parameter **chanNum**: the **MPEGIO2** channel number from 0 ~ totalChans-1
Output Parameter **inputAudioMuted**: true means all the 1 ~ 3 audio **input** stereo channels are muted
Output Parameter **outputChanMuted**: true means the audio **output** channels from MPEG decoder to external speakers and the audio coming from the matching channels' output and connected to this channel's MPEG decoder audio output are both muted.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setAudio6DB**(unsigned long chanNum, bool set6dB);
Function: Set -6dB reduction on input audio signals on an **MPEGIO2** Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **set6dB**: true to reduce audio input by -6dB, false (default) not to set this reduction
Return true for success.
Note: This setting affects all audio input channels simultaneously on this **MPEGIO2** channel.

MPEGIO2_API bool **MPEGIO2_getAudio6DB**(unsigned long chanNum, bool &set6dB);
Function: Get the -6dB reduction setting on input audio signals for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **set6dB**: true to reduce audio input by -6dB, false (default) not to set this reduction
Return true for success.

MPEGIO2_API bool **MPEGIO2_OSDICasAudioADC**(unsigned long chanNum, bool useOSDIC);
Function: Select to Use OSD IC or Preview IC as the Audio Analog-to-Digital Converter (ADC) for MPEG encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **useOSDIC**: True to use OSD IC, false (default) to use Preview IC, as the Audio Analog-to-Digital Converter to digitize input analog audio to send to MPEG Codec IC.
Return true for success.
Note 1: Differences between using OSD IC and Preview IC as Audio ADC:
Using OSD IC:
(1). can have more adjustment on input and output audio gains
(2). can set different mixing audio channels' ratios for MPEG audio encoding
(3). Monitoring output at Stereo Out RCA3(J20)/Stereo Out RCA4(J25) will be identical for left channel

and right channel stereo (mixed the left and right together)

(4). Audio preview/monitoring and Audio MPEG encoding can have different Sampling Rates

(5). Audio Preview Signal of this channel will connect to the audio input pins AIN0/AIN1 on the OSD IC

(6). Audio Monitor Output Source will affect audio encoding (see **MPEGIO2_setAudioMonitorOutSrc**)

(7). Audio from J10(if **chanNum** is Left Channel) or J21(if **chanNum** is Right Channel) on **Breakout Box** will be connected directly to the audio input pins AIN2/AIN3 on the OSD IC, therefore J10/J21 audio will always be digitized by OSD IC to be heard on Audio Loopback Output Ports J20/J25 and sent to MPEG Codec IC for encoding even when this channel's audio input is from its matching channel.

Using Preview IC (SDK default):

(1). have less adjusting input and output audio gain

(2). can not set different mixing audio chans'ratios MPEG encoding audio

(3). Monitoring output at Stereo Out RCA3(J20)/Stereo Out RCA4(J25) can have different left channel and right channel stereo signals same as the input from external sockets

(4). Audio preview and Encoding must have the same Sampling Rate.

Note 2: The Preview IC is always used to play audio heard from PC speakers (Audio Preview) regardless which IC is used as Audio ADC to send digitized audio to MPEG Codec IC

Note 3: When using Preview IC to preview audio on PC via DMA, 44.1KHz and 48KHz sampling rates cause some background noise when audio pitch is high on PC speakers which is not heard when using 32KHz as sampling rate, although this noise will not be present in the encoded MPEG file/stream. Using 32KHz to preview and using OSD IC to be AADC will avoid this preview-only noise.

MPEGIO2_API bool **MPEGIO2_getOSDICasAudioADC**(
unsigned long chanNum,
bool &useOSDIC);

Function: Get the status of using OSD IC or Preview IC as the Audio Analog-to-Digital Converter for MPEG encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **useOSDIC**: True to use OSD IC, false (default) to use Preview IC, as the Audio Analog-to-Digital Converter to digitize input analog audio to send to MPEG Codec IC.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICAudioSamplingRate**(unsigned long chanNum, byte sr);

Function: Set Audio Sampling Rate when using OSD IC as the Audio Analog-to-Digital Converter for MPEG encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **sr**: 0: 32kHz, 1: 44.1kHz , 2: 48kHz
Return true for success.

MPEGIO2_API bool **MPEGIO2_getOSDICAudioSamplingRate**(unsigned long chanNum, byte &sr);

Function: Get Audio Sampling Rate when using OSD IC as the Audio Analog-to-Digital Converter for MPEG encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **sr**: 0: 32kHz, 1: 44.1kHz , 2: 48kHz
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICAudioInGain**(unsigned long chanNum,
unsigned char AINNum,
unsigned char gain);

Function: Set amplifier gain for audio mono sub-channel input AIN0 ~ AIN3 on the OSD IC
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **AINNum**: 0 = AIN0(Left sub-channel of Audio Line-In 1 or Mic),
1 = AIN1(Right sub-channel of Audio Line-In 1 or Mic),
2 = AIN2(Left sub-channel of Audio Line-In 2),
3 = AIN3(Right sub-channel of Audio Line-In 2).
Input Parameter **gain**: 0 ~ 15: gain applied to input signals
0: 0.25
1: 0.31
2: 0.38
3: 0.44(hardware reset default)
4: 0.50
5: 0.63
6: 0.75

7: 0.88
8: 1.00
9: 1.25
10: 1.50
11: 1.75
12: 2.00
13: 2.25
14: 2.50
15: 2.75

Return true for success.
Note 1: This gain will not affect audio heard on PC speakers during audio preview.
Note 2: When OSD IC is used as Audio ADC, this gain also affects the audio encoded in MPEG file/stream.
Note 3: High gain values can cause corrupted audio in MPEG encoding.

MPEGIO2_API bool **MPEGIO2_getOSDICAudioInGain**(unsigned long chanNum,
unsigned char AINNum,
unsigned char &gain);

Function: Get amplifier gain for audio mono sub-channel input AIN0 ~ AIN3
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **AINNum**: Same as in function **MPEGIO2_setOSDICAudioInGain**
Output Parameter **gain**: Same meaning as in function **MPEGIO2_setOSDICAudioInGain**
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICAudioOutGain** (unsigned long chanNum,
unsigned char gain);

Function: Set amplifier gain for the audio output socket J20/J25 (Audio Loopback Output) when the OSD IC is Audio ADC
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **gain**: Same meaning as in function **MPEGIO2_setOSDICAudioInGain**
Return true for success.
Note: This gain will not affect audio heard on PC speakers during audio preview, nor audio encoded in MPEG file/stream.

MPEGIO2_API bool **MPEGIO2_getOSDICAudioOutGain**(unsigned long chanNum,
unsigned char &gain);

Function: Get amplifier gain for the audio output socket J20/J25 (Audio Loopback Output) when the OSD IC is Audio ADC
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **gain**: Same meaning as in function **MPEGIO2_setOSDICAudioInGain**
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOSDICAudioOutChan**(unsigned long chanNum,
unsigned char AINNum);

Function: Set OSD IC's Audio DAC (Digital to Analog Converter)'s Input Signal Source as from one of AIN0~AIN3 or as mixing all of them
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **AINNum**: 0 = AIN0, 1 = AIN1, 2 = AIN2, 3 = AIN3, other values: Mixing AIN0~AIN3:
AINNum0~AINNum3 have the same meanings as in function **MPEGIO2_setOSDICAudioInGain**
Return true for success
Note: OSD IC's Audio DAC's Output is at the Audio loopback Output Sockets J20/J25 on the Breakout Box

MPEGIO2_API bool **MPEGIO2_getOSDICAudioOutChan**(unsigned long chanNum,
unsigned char &AINNum);

Function: Test OSD IC's Audio DAC (Digital to Analog Converter)'s Input Signal Source if from one of AIN0~AIN3 or is mixing all of them
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **AINNum**: 0 = AIN0, 1 = AIN1, 2 = AIN2, 3 = AIN3, other values: Mixing AIN0~AIN3:
AINNum0~AINNum3 have the same meanings as in function **MPEGIO2_setOSDICAudioInGain**
Return true for success
Note: OSD IC's Audio DAC's Output is at the Audio loopback Output Sockets J20/J25 on the Breakout Box.

MPEGIO2_API bool **MPEGIO2_enableOSDICAADC**(unsigned long chanNum, bool enable);
Function: Enable/Disable OSD IC's Audio Analog-to-Digital Converter (ADC) Function

Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **enable**: true (default) to enable, false to disable OSD IC's Audio ADC Function (ability to accept analog audio and output them in digitized form)
Return: True for success
Note 1: This function is only useful when OSD IC is set as the Audio Analog-to-Digital Converter through function "**MPEGIO2_OSDICasAudioADC**".
Note 2: disabling OSD Audio ADC will result in no audio in encoded MPEG file/stream and no sound at the Audio Loopback Output ports on the Breakout Box (J20/J25 socket).

MPEGIO2_API bool **MPEGIO2_getEnableOSDICAADC**(unsigned long chanNum, bool &enable);
Function: Get the Enable/Disable status of OSD IC's Audio Analog-to-Digital Converter Function
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output Parameter **enable**: When the function returns true, Get true (default) for enabling, get false for disabling OSD IC's Audio ADC Function (ability to accept analog audio and output them in digitized form)
Return: True for success

MPEGIO2_API bool **MPEGIO2_OSDICAudioInLevel**(unsigned long chanNum,
unsigned long AINNum,
unsigned short &audLevel,
unsigned short &adjLevel);
Function: Get the current Audio Input Level of one of the 4 Audio Input Pins of the OSD IC
Input Parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input Parameter **AINNum**: 0 ~ 3: the OSD IC's Audio Input Pin, same as in function **MPEGIO2_setOSDICAudioInGain:**
0 = AIN0(Left sub-channel of Audio Line-In 1 or Mic),
1 = AIN1(Right sub-channel of Audio Line-In 1 or Mic),
2 = AIN2(Left sub-channel of Audio Line-In 2),
3 = AIN3(Right sub-channel of Audio Line-In 2).
Output Parameter **audLevel**: the AINNum Pin's current audio level, 0 ~ 511
Output Parameter **adjLevel**: the AINNum Pin's current adjusted audio level, 0 ~ 511
Return: True for success
Note: This function is only meaningful when the OSD IC is currently used as Audio Analog-to-Digital Converter via calling function **MPEGIO2_OSDICasAudioADC**(chanNum, true),
otherwise this function returns false without assiging values to **audLevel** and **adjLevel.**

4.5 Video Preview Functions

Incoming video passing through the A/V Input and OSD processing IC can be live previewed on PC's Screen within one or more resizable windows. Once calling the Video Preview Functions to enable video preview, the **MPEGIO2** SDK will handle the preview window's resize, move, full-screen etc. operations automatically without any help from the application software: although if the application software so wishes it can manipulate the video preview window directly because the video preview window's handle is passed back to the application when the **MPEGIO2_startPreviewWindow** function is called.

Under MS Windows XP and Windows 7 (or above) the Video Preview functions differ in their syntax due to different video overlay APIs used: under Windows XP DirectDraw is used, while under Windows 7 and above DirectX3D is used.

```
MPEGIO2_API bool  MPEGIO2_startPreviewWindow(
                                HWND parentWnd,
                                HWND reportErrorWnd,
                                HWND *videoWnd,
                                RECT *previewWndRect,
                                bool fullScreen,
                                bool showVideo = true,
                                COLORREF keyColor = RGB(255, 128, 128),
                                unsigned long monitorNum = -1,
                                unsigned long chansPerRow = 0,
                                unsigned long interval = 30);
```

Function: This is the **Windows XP** version of **MPEGIO2_startPreviewWindow**():
Start live video preview window on PC screen for all **MPEGIO2** channels.

Input parameter **parentWnd**: the parent window for the video window. If NULL then the video preview window has no parent and will sit directly on desktop window.

Input parameter **reportErrorWnd**: if is a valid window handle, this window receives all messages sent from SDK to the application program. This window does not need to be the same as parameter "parentWnd"

Output parameter **videoWnd**: If not NULL and the function succeeds this will get the handle of the preview video window

Input parameter **previewWndRect**: If not null, indicates the initial screen position of the video preview window, if parameter **fullScreen** is true then this rectangle is the dimension of the video window when it's restored to non-full screen mode.
If this parameter is null then if this is the first time calling **MPEGIO2_startPreviewWindow**, if **initFile** ("**initFile**" is the initialization file such as "**MPEGIO2.ini**", see the parameter "**useInitFile**" in function **MPEGIO2_initSDK** for details) has read back preview window's dimension then that dimension will be used to set the the preview window's dimenstion;
if **initFile** does not exist or is not used or has no preview window dimension then {0, 0, 400, 400} will be used.
If this is not the first time calling this function and this parameter is NULL then desktop window's dimension is used as preview window's initali dimension.

Input parameter **fullScreen**: True to show video in full screen, false to show video as parameter **previewWndRect** indicated.

Input parameter **showVideo**: True to show the video window, false to hide it

Input parameter **keyColor**: The Key Colour used to paint the background of the shown video window: any pixel on the video that has this colour value will be transparent, so choose this colour as a most unlikely used colour in live video, default is pink (RGB=255/128/128)

Input parameter **monitorNum**: Video preview window is created for and video is displayed on this monitor only in the PC: 0 for 1st monitor, 1 for 2nd monitor, 2 for 3rd, ... The default is -1 meaning the SDK will automatically use the primary monitor in the PC if multiple monitors are attached to (possibly multiple) graphics cards. Moving the Video Preview Window to a monitor different from this one will see no video in the window.

Input parameter **chansPerRow**: how many channels will be displayed per row in the video preview window. If 0(default) or > **totalChans** (as returned by **MPEGIO2_initSDK**), then all channels will be displayed on the same row with same height, otherwise **chansPerRow** channels' video will be displayed on each row with equal height and width.

Input parameter interval: time interval in milli seconds to update video display, must be between 1~100, default is 30

Return: true for success.
Note 1: Preview window will automatically handle some mouse-pressings:

- Left mouse Double click: toggle fulle screen mode
- Left mouse Double click with Ctrl-key down: toggle between single channel's video to occupy the full preview window or to display all channels' video inside the preview window

Note 2: Starting Video Preview is un-related to MPEG encoding/decoding operation: video can be encoded/decoded without starting video preview, i.e., even if no video is on PC screen they could still be encoded, recorded, streamed, etc.

MPEGIO2_API bool **MPEGIO2_stopPreviewWindow(RECT *rect = NULL);**

Function: This is **Windows XP** version of **MPEGIO2_stopPreviewWindow**:
Stop live video preview window on PC screen for all **MPEGIO2** channels: Before calling **MPEGIO2_endSDK** this function **Must** be called if video preview is currently on or system crash could happen.
Input parameter **rect**: if not NULL(NULL is default) indicate the window area on desktop window to clear after stopping preview, if this is NULL then the entire desktop window will be refreshed after stopping preview
Return: true for success.

```
MPEGIO2_API bool  MPEGIO2_startPreviewWindow(
        unsigned long chanNum,
        HWND parentWnd,
        HWND reportErrorWnd,
        HWND *appVideoWnd,
        RECT *previewWndRect,
        unsigned int deinterlaceDevIndex,
        bool fullScreen = false,
        bool showVideo = true,
        COLORREF keyColor = RGB(255, 128, 128),
        unsigned long interval = 30,
        ULONG previewDMAWidth = 600,
        ULONG previewDMAHeight = 900,
        ULONG *deinterlace = 0,
        LONG previewAlpha = -1,
        bool topMost = false,
        bool allInOneWindow = false,
        unsigned long chansPerRow = 0,
        bool oneSurface = false,
        unsigned long previewDMARamMode = 0,
        unsigned long monitorNum = -1);
```

Function: This is **Windows 7** or above version of **MPEGIO2_startPreviewWindow**():
Start live video preview window for one or all **MPEGIO2 channels** on PC screen

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number, this parameter will be ignored if parameter **allInOneWindow** or **oneSurface** is true
Input parameter **parentWnd**: the parent window for the video window, If NULL then the video preview window has no parent and will sit directly on the desktop window

Input parameter **reportErrorWnd**: if is a valid window handle, this is the window receving error report messages sent from SDK to application program

Output parameter **appVideoWnd**: If not NULL and this function succeeds this parameter will get the handle of the preview video window

Input parameter **previewWndRect**: the screen position of the video window, if NULL, the desktop window's rect is used

Input parameter **deinterlaceDevIndex**: which Direct3D De-interlace algorithm to use: from 0 upto the (numOfDeinterlaceDev - 1) where numOfDeinterlaceDev is returned by calling **MPEGIO2_getD3DCaps**() function. In particular the parameter **defaultBobDevIndex** returned by **MPEGIO2_getD3DCaps** can be used here**.**

Input parameter **fullScreen**: True to show video in full screen, false will show video as previewWndRect indicated

Input parameter **showVideo**: true to show the video window, false to hide it

Input parameter **keyColor**: the Key Colour used to paint the background of the shown video window: any pixel on the video that has this colour value will be transparent, so choose this colour as a most unlikely used colour in live video, default is pink (RGB=255/128/128)

Input parameter **interval**: interval time in millie-second to update a video frame on PC screen during video preview process, default is 30

Input parameter **previewDMAWidth/previewDMAHeight**: this channel's preview video's width/height inside the entire video preview window, in unit of pixels.

Input parameter **deinterlace**: If NULL, all channels will use deinterlace method 1 (copy odd lines to even lines).
If not NULL, must point to an integer array with at least **totalChans** members each has value:
0: SDK does not do Deinterlace when displaying video on PC screen,
1: SDK does Deinterlace by copying odd lines to even lines,
2: SDK does Deinterlace by copying even lines to odd lines

Input parameter **previewAlpha**: the alpha value used to display the video, 0xFFFFFFFF(-1) means full visibility

Input parameter **topMost**: true to set the preview window as top most, false set it to bottom

Input parameter **allInOneWindow**: If true then all channels will be shown inside one preview window, This parameter is ignored if parameter oneSurface is true.

Input parameter **chansPerRow**: only meaningful if allInOneWindow is true or oneSurface is true and totalChans > 1:
how many channels' video preview will be arranged on one row inside the preview window
If this parameter is < 1 or > totalChans then totalChans will be assumed which means all channels will be arranged on one row.

Input parameter **oneSurface**: Use only one Direct3D Surface to preview all MPEGIO2 channels' video so that CPU usage is low when multiple **MPEGIO2** channels are previewed.

Input parameter **previewDMARamMode:** Memory Mode to do Live Video Preview DMA on PC screen:
1: Use Video Memory as DMA RAM and lock the D3D surface when updating video frame on screen,
2: Use System Memory as DMA RAM and unlock the D3D surface when updating video frame on screen,
Mode 1 is suitable for ATI graphics card and latest NVideo graphics cards,
Mode 2 is less efficient but works on Intel Graphics Chipsets(Q67 etc) and some older NVidia Graphics Card GForce 8400 etc);
0(default) or any other value: SDK auto decides to use Mode 1 or Mode 2 according to the capability of the Graphics card to do Colour Space Conversion from YUV to RGA: if the graphics card can do such a colour space conversion then use Mode 1, otherwise use Mode 2.

Input parameter **monitorNum**: Video preview window is created for and video is displayed on this monitor only on the PC:
0 for 1st monitor, 1 for 2nd monitor, 2 for 3rd, ..., note the default is -1 meaning the SDK will automatically use the primary monitor in the PC if multiple monitors are attached to (possibly multiple) graphics cards.
Moving the Video Preview Window to a monitor different from this one will see no video in the window.
Return: true for success.

MPEGIO2_API bool **MPEGIO2_stopPreviewWindow**(unsigned long chanNum,
RECT *rect = NULL);

Function: This is **Windows 7** or above version of **MPEGIO2_stopPreviewWindow**:
Stop live video preview window on PC screen for a channel.

Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **rect**: if not NULL(NULL is default) indicate the window area on desktop window to clear after stopping preview, if this is NULL then the entire desktop window will be refreshed after stopping preview
Return: true for success.

MPEGIO2_API unsigned long **MPEGIO2_getTotalMonitors**(unsigned long &primaryMonitorNum);
Function: Return the total number of attached monitors in the PC.
Output Parameter **primaryMonitorNum**: the primary monitor number, will be between 0 and total monitor number – 1.
Note: This function can be called any time without calling **MPEGIO2_initSDK**() first.

MPEGIO2_API bool **MPEGIO2_getD3DCaps**(unsigned long monitorNum,
unsigned int &numOfDeinterlaceDev,
int &defaultBobDevIndex,
bool &allowHardwareOverlay,
unsigned int &maxOverlayWidth,
unsigned int &maxOverlayHeight,
bool &overlayStretchX,
bool &overlayStretchY,
bool &canDoCSC);
Function: for Windows 7 or above: retrieve the Direct3D capacities that can be used as parameter values for video preview, in particular the returned value **defaultBobDevIndex** can be used in function **MPEGIO2_startPreviewWindow.**
Input Parameter **monitorNum**: the current PC Monitor number, valid is 0 ~ **MPEGIO2_getTotalMonitors**()-1. If a value >= **MPEGIO2_getTotalMonitors**() is supplied the SDK will use the primary monitor.
Output Parameter **numOfDeinterlaceDev:** number of DXVA Deinterlace devices available for the graphcs card hardware
Output Parameter **defaultBobDevIndex**: index of the **Bob** Deinterlace device inside the **numOfDeinterlaceDev** DXVA Deinterlace devices, this can be used as parameter **deinterlaceDevIndex** when calling function **MPEGIO2_startPreviewWindow**
Output Parameter **allowHardwareOverlay**: If the graphics card allow hardware overlay
Output Parameter **maxOverlayWidth:** maximum overlay width in pixels
Output Parameter **maxOverlayHeight:** maximum overlay height in pixels
Output Parameter **overlayStretchX:** if to allow overlay stretch horizontally
Output Parameter **overlayStretchY:** if to allow overlay stretch vertically
Output Parameter **canDoCSC:** if the graphics card can do YUV(UYVY) to RGB(X8B8G8R8) colour space conversion.
Return true for success.
Note: for Windows XP this function does nothing and returns true.

MPEGIO2_API bool **MPEGIO2_freezeVideoPreview**(unsigned long chanNum = 0,
bool freeze = true);
Function: Temporarily freeze/restore video display inside video preview window PC screen for all **MPEGIO2** channels (Windows XP) or for one **MPEGIO2** Channel (Windows 7) that are previewing video
Input parameter **chanNum**: Ignored for Windows XP, valid for Windows 7: must be between 0 ~ totalChans-1
Input parameter **freeze**: true to freeze, false to restore video preview display
Return: true for success
Note: freezing video preview does not affect video being recorded/streamed and sent to callback function. Nor will it affect the still image capture.

MPEGIO2_API bool **MPEGIO2_isPreviewing**(unsigned long chanNum = 0,
HWND *wnd = NULL,
bool *freeze = NULL
bool *oneWnd = NULL);
Function: Test if a channel is previewing video.
Input Parameter **chanNum**: must be between 0 ~ totalChans - 1 for Windows 7 or above, ignored for Windows XP since it always previews all channels together.
Output Parameter **wnd**: if supplied as not NULL and this fuction returns true then will receive the preview window handle
Output Parameter **freeze**: if not NULL will receive the freeze status of the video preview: true means preview is frozen, false means preview is not frozen. Only be set if the channel is previewing video.
Output Parameter **oneWnd:** if not NULL, and if the channel is previewing, this get true if all channels' preview are inside one video window: for WinXP, this is always true, for Win7 or above, this can be true or false depending on the value of parameter "**allInOneWindow**" passed to function **MPEGIO2_startPreviewWindow.**

Return: true if this channel is previewing video.

MPEGIO2_API bool **MPEGIO2_restartVideoPreview**(unsigned long chanNum);
Function: Stop then re-start one channel's preview on screen
Input parameter **chanNum**: must be between 0 ~ totalChans-1
Return true for success
Note 1: The video preview must be in progress when calling this function
Note 2: Stop/restart preview might be necessary in situations such as input signal changing format (PAL/NTSC switch) to rectify corrupted preview image.

MPEGIO2_API bool **MPEGIO2_getSystemParams**(unsigned long &chansPerRow, unsigned long &interval);
Function: Get system parameters as passed over from application program to **MPEGIO2_startPreviewWindow** function.
Output parameter **chansPerRow**: as passed over to function **MPEGIO2_startPreviewWindow**:
if preview is in progress, this returns the number of channels per horizontal row displayed.
if preview is not in progress or never started, this is the channel per row value last time the preview was in progress (if preview has never been started since **MPEGIO2** SDK was invoked this is the channel per row value read back from **initFile** MPEGIO.ini which is channel per row when the last time the preview was in progress)
Output parameter **interval**: time interval in milli second unit to update video display, as passed over to function **MPEGIO2_startPreviewWindow**
Return true if succeed.

MPEGIO2_API bool **MPEGIO2_showPreviewWindow**(unsigned long chanNum, bool show);
Function: Make video preview window visible or hidden on one channel or all channels
Input parameter **chanNum**: Only meaningful on Windows 7 when parameters **allInOneWindow** and **oneSurface** passed to last call of function **MPEGIO2_startPreviewWindow** were FALSE (therefore parameter **chanNum** to **MPEGIO2_startPreviewWindow** was NOT ignored).
On WindowsXP this function has effects on all **MPEGIO2** channels so **chanNum** is ignored.
On Windows 7 when this parameter is meaningful it must be between 0 ~ totalChans-1.
Input parameter **show**: true to show video preview window, false to hide it (but not distroy it).
Return true if succeed.
Note: when calling this function the video preview window must have been created by calling function **MPEGIO2_startPreviewWindow**, otherwise this function returns false without any effect.

MPEGIO2_API bool **MPEGIO2_setSingleChanPreview**(unsigned long chanNum);
Function: Switch to single channel or all channel preview mode inside the preview window
Input parameter **chanNum**: If this is < totalChans then make the video preview window to display only this channel's Video, otherwise display all **totalChans** channels' video inside the preview window with **chansPerRow** channels' video on each row, where **chansPerRow** is the parameter passed to **MPEGIO2_startPreviewWindow**(0=all channels on one row).
Return true if succeed.
Note 1: This function can only be called after a successful call to **MPEGIO2_startPreviewWindow.**
Note 2: If **totalChans** is < 2 this function returns false and does nothing.
Note 3: In Windows 7 or above: if the **MPEGIO2_startPreviewWindow** function is started with parameter **allInOneWindow**=false then this function returns false and does nothing.
Note 4: If 0 <= chanNum < **totalChans** and the preview is already in this channel's preview mode this function does nothing and return false.
Note 5: If chanNum >= **totalChans** and the preview is already in all channels' preview mode this function does nothing and returns false.

MPEGIO2_API unsigned long **MPEGIO2_getSingleChanPreview**(void);
Function: Get the channel's number that is doing single channel preview inside the Video Preview Window: the channel whose video is occupying the full client area of the Video Preview Window in front of all other channels' video.
Return value: if the preview is in single channel preview mode then return the channel number being previewed inside the video window(this channel's video preview is fully on the client area of the Video Preview Window), else return **totalChans** which means all channels are being displayed inside the video preview window.
Note 1: If there is only 1 **MPEGIO2** channel availabe in the PC then this function always returns 0 if preview is on.
Note 2: Under Windows 7 or above if preview is not on all channel preview mode (parameter **allInOneWindow** was false when calling **MPEGIO2_startPreviewWindow**) then this function returns -1.
Note 3: If video preview is not on (inc. when no channel is previewing video) this function returns -1.

MPEGIO2_API bool **MPEGIO2_setVideoPreviewSrcClip**(unsigned long chanNum,
long left,
long top,
long right,
long bottom);

Function: Set the Video Preview Source Clipping Pixels on an **MPEGIO2** Channel
Input parameter **chanNum**: For Win7 and above, 0 ~ totalChans-1: the channel number. for WinXP: this is ignored.
Input parameter **left**: the pixels to include (when **left** is negative) or exclude (when left is positive) from the left edge of the incoming video
Input parameter **top** : the pixels to include (when **top** is negative) or exclude (when top is positive) from the top edge of the incoming video
Input parameter **right**: the pixels to include (when **right** is positive) or exclude (when right is negative) from the right edge of the incoming video
Input parameter **bottom**: the pixels to include (when bottom is positive) or exclude (when bottom is negative) from the bottom edge of the incoming video
Return true for success
Note 1: To preview only a smaller portion of the incoming video's raw frame, set parameter **left** and **top** to positive, **right** and **bottom** to negative
Note 2: The default is: **left** = **top** = **right** = 0, **bottom** = -12.
Note 3: This clipping setting will not affect still image captured on the channel
Note 4: The values set by this function will only be effective when channel preview is turned on.

MPEGIO2_API bool **MPEGIO2_getVideoPreviewSrcClip**(
unsigned long chanNum,
long &left,
long &top,
long &right,
long &bottom);

Function: Get the Video Preview Source Clipping Pixels on an **MPEGIO2** Channel
Input parameter **chanNum**: For Win7 and above, 0 ~ totalChans-1: the channel number. for WinXP: this is ignored.
Output parameter **left**: the pixels to include (when left is negative) or exclude (when left is positive) from the left edge of the incoming video
Output parameter **top** : the pixels to include (when top is negative) or exclude (when top is positive) from the top edge of the incoming video
Output parameter **right**: the pixels to include (when right is positive) or exclude (when right is negative) from the right edge of the incoming video
Output parameter **bottom**: the pixels to include (when bottom is positive) or exclude (when bottom is negative) from the bottom edge of the incoming video
Return true for success
Note: The default is: **left** = **top** = **right** = 0, **bottom** = -12

MPEGIO2_API bool **MPEGIO2_getVideoWndRect**(unsigned long chanNum, RECT &rect);

Function: Return the Video Preview Window rectangle for a channel if the video preview window is created
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **rect**: return the video preview window's left, top, right, bottom position on PC's screen in screen co-oridnate if the video window is a top-level window.
If the video window is a child window then the returned **rect** is in the parent window's client area co-ordinates.
Return true for success. If failed, the **rect** content will not be changed.
Note: When the preview window is displaying only one channel's video:
If the totalChans is 1 or the displayed channel number == **chanNum**,
then the entire video preview window's rectangle is returned in parameter "**rect**",
Otherwise (totalChans > 1 and the currently displayed channel number is not the same as parameter "**chanNum**"), false is returned and **rect** is not changed.

MPEGIO2_API unsigned long **MPEGIO2_getChanNumAtCursor**(void);

Function: Return the **MPEGIO2** channel number (0 ~ totalChans - 1) where the current mouse cursor is hovering,
On all error conditions return 0: this includes when the mouse cursor is outside the video window.

MPEGIO2_API bool **MPEGIO2_setManualDeinterlace**(unsigned long chanNum,
ULONG deinterlace,
ULONG conditionalDeinterlace = 0,
ULONG deinterlaceWidthRatio = 50,
ULONG deinterlaceHeightRatio = 50);

Function: For Windows 7: set up the manual(host based software) deinterlace mode for a channel video preview,
For WinXP: does nothing(always returns true).
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **deinterlace**: 1 for enabling unconditional deinterlace, 0 for disabling unconditional deinterlace
Input parameter **conditionalDeinterlace**: setup conditional deinterlacing when parameter **deinterlace** is 0:
0: not to do conditional deinterlace (default setting)
1: odd-field deinterlace only when preview window width exceeds
(deinterlaceWidthRatio/100) * desktopWindow Width
or when preview window height exceeds (deinterlaceHeightRatio/100) * desktopWindow Height
2: even-field deinterlace only when preview window height exceeds
(deinterlaceHeightRatio/100) * desktopWindow Height
or when preview window width exceeds (deinterlaceWidthRatio/100) * desktopWindow Width
4: do odd-field deinterlace when preview window is in focus (odd lines are copied to even lines)
8: do even-field deinterlace when preview window is in focus(even lines are copied to odd lines).

Input parameter **deinterlaceWidthRatio**, **deinterlaceHeightRatio**: ratios used in calculating if the conditions are met in the above description
Return: True on success. If the channel is not currently previewing video this function returns false and does nothing.
Note: If parameter **deinterlace** is 1 then **conditionalDeinterlace's** value becomes useless since then deinterlace is always on.

MPEGIO2_API bool **MPEGIO2_getManualDeinterlace**(unsigned long chanNum,
ULONG &deinterlace,
ULONG &conditionalDeinterlace,
ULONG &deinterlaceWidthRatio,
ULONG &deinterlaceHeightRatio);

Function: For Windows 7: Retrieve the manual(host based software) deinterlace mode for a channel's preview video.
For WinXP: does nothing(always returns true).
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **deinterlace**: 0(Default): No Deinterlace is done on video preview;
1: Odd Lines Are Copied to Even Lines in each Video Frame;
2: Even Lines are copied to Odd Lines in each Video Frame.
Output parameter **conditionalDeinterlace**: conditional deinterlacing method as set up in function **MPEGIO2_setManualDeinterlace**
Output parameter **deinterlaceWidthRatio**, **deinterlaceHeightRatio**:
ratios used in calculating if the deinterlacing conditions are met.
Return: True on success. If the channel is not currently previewing video this function returns false and does not set any value to output parameters

MPEGIO2_API bool **MPEGIO2_setBrightPorS**(unsigned long chanNum,
unsigned char bright,
bool preview);

Function: Set the Video Preview or Still Image Capture Luminance Brightness on an **MPEGIO2** Channel
Video Preview Brightness only affects on screen video preview's Brightness;
Still Image Brightness only affects captured image file's Brightness.
This setting will not affect MPEG recording's colour.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **bright**: the new Brightness value, within 0 ~ 255, default is 0x80
0x00: all black
0x80: nominal value
0xFF: all white
Input parameter **preview**: true to set video preview brightness, false to set still image capture brightness
Return true for success

Note: On SDK Start up, if this function has never been called the default Brightness value will be used.

MPEGIO2_API bool **MPEGIO2_getBrightPorS**(unsigned long chanNum,
unsigned char &bright,
bool preview);

Function: Get the current Video Preview or Still Image Capture Luminance Brightness on a Channel
Video Preview Brightness only affects on screen video preview's Brightness;
Still Image Brightness only affects captured image file's Brightness.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **bright**: the retrieved current Brightness value
Input parameter **preview**: true to get video preview Brightness, false to get still image capture Brightness
Return true for success

MPEGIO2_API bool **MPEGIO2_setContrastPorS**(unsigned long chanNum,
unsigned char contrast,
bool preview);

Function: Set the Video Preview or Still Image Capture Luminance Contrast on a Channel
Video Preview Contrast only affects on screen video preview's contrast;
Still Image Contrast only affects captured image file's contrast.
This setting will not affect MPEG recording's colour.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **contrast**: the new contrast value, within 0 ~ 255, Default is 0x40
0x00: flat
0x40: nominal value
0xFF: four-fold contrast
Input parameter **preview**: true to set video preview contrast, false to set still image capture contrast
Return true for success
Note: On SDK Start up, if this function has never been called the default Contrast value will be used.

MPEGIO2_API bool **MPEGIO2_getContrastPorS**(unsigned long chanNum,
unsigned char &contrast,
bool preview);

Function: Get the current Video Preview or Still Image Capture Luminance Contrast on an **MPEGIO2** Channel
Video Preview Contrast only affects on screen video preview's contrast;
Still Image Contrast only affects captured image file's contrast.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **contrast**: the retrieved current contrast value, within 0 ~ 255
Input parameter **preview**: true to get video preview contrast, false to get still image capture contrast
Return true for success

MPEGIO2_API bool **MPEGIO2_setSaturationPorS**(unsigned long chanNum,
unsigned char saturation,
bool preview);

Function: Set the Video Preview or Still Image Capture Chrominance Saturation on an **MPEGIO2** Channel
Video Preview Saturation only affects on screen video preview's Saturation;
Still Image Saturation only affects captured image file's Saturation.
This setting will not affect MPEG recording's colour.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **saturation**: the new Saturation value, within 0 ~ 255, default is 0x40:
0x00: no colour
0x40: nominal value
0xFF: four-fold saturation
Input parameter **preview**: true to set video preview Saturation, false to set still image capture Saturation
Return true for success
Note: On SDK Start up, if this function has never been called the default Contrast value will be used.

MPEGIO2_API bool **MPEGIO2_getSaturationPorS**(unsigned long chanNum,
unsigned char &saturation,

bool preview);

Function: Get the current Video Preview or Still Image Capture Chrominance Saturation on an **MPEGIO2** Channel
Video Preview Saturation only affects on screen video preview's Saturation;
Still Image Saturation only affects captured image file's Saturation.

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Output parameter **saturation**: the retrieved current saturation value, within 0 ~ 255

Input parameter **preview**: true to get video preview saturation, false to get still image capture saturation

Return true for success

4.6 Preview Text Functions

When Video Preview Window is started, **MPEGIO2** SDK allows 2 lines of text up to 80 ASCII characters (no Unicode support) to overlay on top of the live video, these functions control the properties and placement of these two text overlays (referred to as text1 and text2). These 2 overlay text lines will not appear on the encoded MPEG video nor on the video **Loopback** output ports: they only appear on PC screen inside the Video Preview Window, thus is mainly used for status display etc. Note Windows XP and Windows 7 or above have different function syntax for preview text display.

```
MPEGIO2_API bool  MPEGIO2_setupPreviewText(
                                unsigned long chanNum,
                                unsigned int width = 24,
                                int height = 24,
                                unsigned int weight = FW_NORMAL,
                                unsigned int italic = 0,
                                COLORREF colour = RGB(255, 0, 0),
                                LOGFONT *lFont = NULL,
                                int bkMode = TRANSPARENT,
                                COLORREF bcolour = RGB(255, 255, 255));
```

Function: This is Windows XP version of **MPEGIO2_setupPreviewText**:
Setup Text Display inside Video Preview Window on PC Screen for an **MPEGIO2** channel.
Input Parameter **chanNum**: must be between 0 ~ totalChans - 1
Input Parameter **width**: the width of the font used to display text, default is 24, cannot be 0
Input Parameter **height**: the height of the font used to display text, default is 24, cannot be 0
Input Parameter **weight**: the weight of the font used to display text, default is 400, within 0~1000, bigger value is heavier
Input Parameter **colour** : text colour in RGB format
Input Parameter **lFont**: If not NULL this is used to override the default font used to display the text string,
If not null then parameters **width**, **height** & **weight** values will be overridden by this font's values
Input Parameter **bkMode** : backgraound mode: either TRANSPARENT(1) or OPAQUE(2)
Input Parameter **bcolour**: background colour when **bkMode** is OPAQUE
Return: true for success
Note: This function only set up preview text's display format, to start or stop preview text display, call function
MPEGIO2_startPreviewText1 or **MPEGIO2_setupPreviewText2**

```
MPEGIO2_API bool MPEGIO2_getPreviewText( unsigned long chanNum,
                                         bool &displayText,
                                         char **textStr,
                                         int &x,
                                         int &y,
                                         LOGFONT **lFont,
                                         COLORREF &colour,
                                         int &bkMode,
                                         COLORREF &bcolour);
```

Function: This is Windows XP Version of **MPEGIO2_getPreviewText**():
Retrieve Text Display parameterts inside Video Preview Window on PC Screen for a channel
Input Parameter **chanNum**: must be between 0 ~ totalChans - 1
Output Parameter **displayText**: If getting true the following parameters are used to display text inside video preview window,
if receiving false no text is to be displayed inside video preview window.
Output Parameter **textStr**: If not NULL, will point to the text string to be displayed as the 1st text line (text1).
If NULL, this parameter has no effect.
Output Parameter **x**: will get the text1 starting horizontal position inside video frame in pixel unit
Output Parameter **y**: will get the text1 starting vertical position inside video frame in pixel unit
Output Parameter **lFont**: If not NULL, this is used to get the font used to display the text string.
If is NULL this parameter has no effect.
Output Parameter **colour** : will get the text colour
Output Parameter **bkMode** : will get the backgraound mode: either TRANSPARENT(1) or OPAQUE(2)
Output Parameter **bcolour**: will get the background colour when bkMode is OPAQUE
Return: true on success.

```
MPEGIO2_API bool  MPEGIO2_setupPreviewText(
                                        unsigned long chanNum,
                                        unsigned int width = 10,
                                        int height = 10,
                                        unsigned int weight = FW_NORMAL,
                                        unsigned int italic = 0,
                                        unsigned long alpha = 255,
                                        unsigned long red = 255,
                                        unsigned long green = 0,
                                        unsigned long blue = 0,
                                        char *typeFace = "Times New Roman");
```

Function: This is For Windows 7 or Above **MPEGIO2_setupPreviewText**:
Setup the 1st Text Display Line(text1) inside Video Preview Window on PC Screen for a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **width**: font width in pixel unit
Input Parameter **height**: font height in pixel unit
Input Parameter **weight**: font weight, must be within 0 ~ 1000, default is 400, larger value mean heavier print
Input Parameter **italic**: font italic: non-zero for italic, 0 for normal
Input Parameter **alpha**: display alpha,within 0 ~ 255
Input Parameter **red:** display colour's red component, within 0 ~ 255
Input Parameter **green**: display colour's green component, within 0 ~ 255
Input Parameter **blue**: display colour's blue component, within 0 ~ 255
Input Parameter **typeFace**: If not NULL, must point to a valid font's typeface name
Return: true on success
Note: This function only set up preview text's display format, to start or stop preview text display, call function
MPEGIO2_startPreviewText1 or **MPEGIO2_setupPreviewText2**.

```
MPEGIO2_API bool  MPEGIO2_getPreviewText(unsigned long chanNum,
                                         char **textStr,
                                         int &x,
                                         int &y,
                                         int &width,
                                         int &height,
                                         unsigned int &weight,
                                         unsigned int &italic,
                                         unsigned long &alpha,
                                         unsigned long &red,
                                         unsigned long &green,
                                         unsigned long &blue,
                                         char **typeFace = 0);
```

Function: This is Windows 7 or Above version of **MPEGIO2_getPreviewText**():
Retrieve the preview text display parameters for an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **textStr**: If not NULL, must be a text string of at least 81-bytes long to hold the displayed string as the 1st text display line (text1) in the form of a null-terminated string.
If a non-NULL **textStr** returns no character then no text will be displayed
Output Parameter **x**: text1 starting horizontal position inside video frame in pixel unit
Output Parameter **y**: text1 starting vertical position inside video frame in pixel unit
Output Parameter **width**: font width
Output Parameter **height**: font height
Output Parameter **weight**: font weight
Output Parameter **italic**: font italic
Output Parameter **alpha**: display alpha
Output Parameter **red**: display colour's red component, within 0 ~ 255
Output Parameter **green**: display colour's green component, within 0 ~ 255
Output Parameter **blue**: display colour's blue component, within 0 ~ 255
Output Parameter **typeFace**: If not NULL, will point to the display font's typeface name
Return: true on success.

MPEGIO2_API bool **MPEGIO2_startPreviewText1**(unsigned long chanNum,
char *textStr,
int x = 0,
int y = 0);

Function: Start or stop displaying the 1st preview text (text1) inside video preview window
Input Parameter **chanNum**: channel number, must be between 0 ~ totalChans - 1
Input Parameter **textStr**: The 1st text string to be displayed as a null terminated string with no more than 80 characters(characters exceeding 80 will not be displayed).
If **textStr** is NULL or contains no character then no text will be displayed (current text display will be stopped) for text1, although text2 could still be displayed if function **MPEGIO2_setupPreviewText2** was or is to be called
Input Parameter **x**: text starting horizontal position inside video frame in pixel unit
Input Parameter **y**: text starting vertical position inside video frame in pixel unit
Return true for success
Note: this function is for Windows XP and Windows 7 or above.

MPEGIO2_API bool **MPEGIO2_setupPreviewText2**(unsigned long chanNum,
char *textStr,
int x = 0,
int y = 0);

Function: Setup the 2nd Text Display Line (text2) inside Video Preview Window for a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **textStr**: a null-terminated text string upto 80-bytes long to be displayed.
If textStr is NULL or contains no character then no 2nd text line will be displayed (call this function with textStr as NULL will disable the 2nd text line display)
Input Parameter **x**: 2nd text line starting horizontal position inside video frame in pixel unit
Input Parameter **y**: 2nd text line starting vertical position inside the Video Preview Window in pixel unit.

Return: true for success
Note : The 2nd text display line (text2) will use the font (size, width, height, colour, typeface etc.) defined by function **MPEGIO2_setupPreviewText** to display its characters inside video preview window, therefore this function can only be called after function **MPEGIO2_setupPreviewText** has been called at least once, otherwise this function returns false and has no effect.
By default there is no 2nd text line to be displayed inside video window, until this function is called.

MPEGIO2_API bool **MPEGIO2_getPreviewText2**(unsigned long chanNum,
char **textStr,
int &x,
int &y);

Function: Retrieve the 2nd Text Display Line information.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **textStr**: If not NULL, must be a string at least 81-bytes long to hold 2nd text line (text2) to be displayed.
If textStr returns no character then no 2nd text line is displayed.
If this is NULL then no text is returned to caller.
Output Parameter **x**: text starting horizontal position inside video frame in pixel unit
Output Parameter **y**: text starting vertical position inside video frame in pixel unit
Return true for success.

MPEGIO2_API bool **MPEGIO2_isPreviewTextOn**(unsigned long chanNum,
char **text1 = 0,
char **text2 = 0);

Function: Test if text display is on inside Video Preview Window on PC Screen for a channel
Input Parameter **chanNum**: channel number, must be between 0 ~ totalChans - 1.
Output Parameter **text**: If not NULL and this channel is displaying preview text, this will point to the 1st text line being displayed on PC screen for this channel
Output Parameter **text2**: If not NULL and this channel is displaying the 2nd preview text, this will point to the 2nd text line being displayed on PC screen for this channel
Return: true if this channel has preview text being displayed in video window, false if it has no preview text being displayed.

```
MPEGIO2_API bool  MPEGIO2_getPreviewTextFont(  unsigned long chanNum,
                                               unsigned int &width,
                                               int &height,
                                               unsigned int &weight,
                                               unsigned int &italic,
                                               COLORREF &colour,
                                               char *typeFace = 0);
```

Function: Return the video preview text display parameters for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **width**: the font's width in logical unit
Output parameter **height**: the font's height in logical unit
Output parameter **weight**: the font's weight: ranges are 0 ~ 1000, bigger value means bolder text displayed
Output parameter **italic**: non-zero for italic, 0 for normal
Output parameter **colour**: The colour of the font
Output parameter **typeFace**: a string of at least 32 characters long to hold the font's type face.
If this is null no type face name will be returned.
Return true for success. If failed, the output parameters' content will not be changed.
Note: For WInXP, the returned parameters are components of the structure "LOGFONT".
For Win7 or above, the returned parameters are components of structure "D3DXFONT_DESC".

4.7 Audio Preview Functions

Audio Preview (or Monitoring) is to play the input audio on PC's speakers. **MPEGIO2** allows audio coming from multiple channels on its PCB to play their incoming audio on PC's speakers simultaneously (or separately), this preview (monitoring) also has volume control and can be set to silence.

```
MPEGIO2_API bool  MPEGIO2_audioMonitorStart(unsigned long chanNum,
                                             bool startAudioMonitor,
                                             unsigned long sampleRateEncode = 1,
                                             unsigned long sampleRatePreview = 0,
                                             unsigned long buffNumPlay = 4,
                                             unsigned long buffNumDMA = 6,
                                             bool useWaveOutAPI = true,
                                             bool directLoop = false,
                                             char *savePCMFile = 0);
```

Function: Start or stop audio preview(monitoring) on PC for a channel

Input Parameter **chanNum**: the MPEGIO2 channel number from 0 ~ totalChans-1

Input Parameter **startAudioMonitor**: True to start audio monitoring, false to stop it.
When this is false all the following parameters will be ignored

Input Parameter **sampleRateEncode**: 0 or 1: 32KHz, 2: 44.1KHz, 3: 48KHz,
audio sampling rate used to encode MPEG audio

Input Parameter **sampleRatePreview**: the audio sampling rate used for PC's playback to its speakers. If less then 100 or > 200000 then this is changed to be the same as parameter "**sampleRateEncode**", otherwise this value is the actual audio sampling rate value in Hz used for PC's sound preview which could be diffrent from the sampling rate used to encode MPEG audio as parameter "**sampleRateEncode**" indicates if OSD IC is used as Audio ADC(see function "**MPEGIO2_OSDICasAudioADC"**).
Note if this value is to indicate actual sampling rate as Hz used on PC speakers' playback, it must be within [DSBFREQUENCY_MIN(100), DSBFREQUENCY_MAX(200000)]: Increasing/decreasing this value changes the perceived pitch of the audio heard on PC speakers, but does not affect the format of the audio data nor affect MPEG encoding result.

Input Parameter **buffNumPlay**: number of buffers to queue up audio chunks read from audio preview device, must be >= 2(default is 4) and <= 128; bigger values mean slower start but smoother audio preview

Input Parameter **buffNumDMA**: number of buffers to queue up audio chunks used inside audio preview device before passing over to SDK, must be >= buffNumPlay(default is 6) and <= 128

Input Parameter **useWaveOutAPI**: true(Default) to use Windows' low-level WaveOutXXX API to monitor audio, false to Use Windows' DirectSound API to monitor sound(on each **MPEGIO2** card using DirectSound is valid on one channel only).

Input Parameter **directLoop**: If true, directly wait and loop over audio data coming from Video preview IC; if false, read, play and wait for the end of the previous play before reading more data from the Video Preview IC. This is for debugging only, application software should always use false for this parameter.

Input Parameter **savePCMFile**: If this parameter point to a valid file name inc. path and extension ".pcm" then the monitored audio will be saved to that file as un-compressed PCM file.

Return true for success.

```
MPEGIO2_API bool  MPEGIO2_audioMonitorIsOn(unsigned long chanNum, bool &silent);
```

Function: Return true if this channel is currently doing audio preview/monitoring.

Input Parameter **chanNum**: the channel number from 0 ~ totalChans-1

Output parameter **silent**: If this is true and function returns true then the monitored audio is not heard on PC speakers (audio preview paused)

```
MPEGIO2_API bool  MPEGIO2_audioMonitorSetSilent(unsigned long chanNum, bool silent);
```

Function: If the channel is monitoring audio, set the audio monitoring's silent value (true or false):

setting to true menas no sound can be heard although audio data is still being pulled out of the MPEGIO2 card
Input Parameter **chanNum**: the MPEGIO2 channel number from 0 ~ totalChans-1
Input Parameter **silent**: true to set silent, false to resume audio monitoring on PC
Return true for success.
Note: If the channel is not currently previewing audio this function returns false without doing anything.

MPEGIO2_API bool **MPEGIO2_audioMonitorGetStatus**(unsigned long chanNum,
bool &silent,
unsigned long &sampleRateEncode,
unsigned long &sampleRatePreview,
unsigned long &buffNumPlay,
unsigned long &buffNumDMA,
bool &useWaveOutAPI,
bool &savePCM);

Function: Return the channel's audio preview/monitoring status.
Input Parameter **chanNum**: the **MPEGIO2** channel number from 0 ~ totalChans-1
Output parameter **silent**: true the channel's audio monitoring on PC speakers is currently set as silent
Output Parameter **sampleRateEncode**: as passed to function "**MPEGIO2_audioMonitorStart**":
0 or 1: 32KHz, 2: 44.1KHz, 3: 48KHz, audio sampling rate used to encode MPEG audio
Output Parameter **sampleRatePreview**: as passed to function "**MPEGIO2_audioMonitorStart**": the audio sampling rate used for PC's playback to its speakers. If less then 100 or > 200000 then this is the same as parameter "**sampleRateEncode**",
otherwise this value is the actual audio sampling rate value in Hz used for PC's sound preview which could be diffrent from the sampling rate used to encode MPEG audio as parameter "**sampleRateEncode**" indicates.
Output Parameter **buffNumPlay**: as passed to function "**MPEGIO2_audioMonitorStart**": number of buffers to queue up audio chunks read from audio preview device.
Output ut Parameter **buffNumDMA**: as passed to function "**MPEGIO2_audioMonitorStart**": number of buffers to queue up audio chunks used inside audio preview device before passing over to SDK.
Output Parameter **useWaveOutAPI**: as passed to function "**MPEGIO2_audioMonitorStart**": true (default) using Windows' low level WaveOutXX API to monitoring audio, false using Windows' DirectSound API
Output Parameter **savePCM**: as passed to function "**MPEGIO2_audioMonitorStart**": true the monitored PCM audio is being saved to file name as passed in parameter **savePCMFile** when calling function **MPEGIO2_audioMonitorStart**, default is false.
Note: This function always returns the correct values regardless the channel's audio preview has started or not

MPEGIO2_API bool **MPEGIO2_audioMonitorSetVolume**(unsigned long chanNum, long volume);
Function: Set audio monitoring volume for an MPEGIO2 channel
Input Parameter **chanNum**: the channel number from 0 ~ totalChans-1
Input Parameter **volume**: new volume for audio monitored on PC's speakers. If audio monitoring is using WaveOutXX API(parameter **useWaveOutAPI** is true to function **MPEGIO2_audioMonitorStart**), volume's low-order word contains the left-channel volume setting, and the high-order word contains the right-channel volume setting.
A value of 0xFFFF represents full volume, and a value of 0x0000 is silence.
If audio monitoring is not using WaveOutXX API(parameter **useWaveOutAPI** is false to function MPEGIO2_audioMonitorStart),thus using DirectSound API, the volume is specified in hundredths of decibels (dB):
allowable values are between DSBVOLUME_MAX (0, no attenuation, maximum sound) and DSBVOLUME_MIN (-10000, silence).
Return true for success
Note 1: If the channel is not monitoring audio this function returns false
Note 2: This volume only affects audio heard on PC's speakers, it does not affect audio encoded into MPEG files/streams.

MPEGIO2_API bool **MPEGIO2_audioMonitorMute**(unsigned long chanNum);
Function: Set audio monitoring volume for an MPEGIO2 channel
Input Parameter **chanNum**: the channel number from 0 ~ totalChans-1
Return true for success
Note 1: If the channel is not monitoring audio this function returns false
Note 2: This volume only affects audio heard on PC's speakers, it does not affect audio encoded into MPEG files/streams.
Note 3: This function simply calls function **MPEGIO2_audioMonitorSetVolume** to set minimum volume value.

MPEGIO2_API bool **MPEGIO2_audioMonitorGetVolume**(unsigned long chanNum, long &volume);

Function: Get current audio monitoring volume for a channel.
Input Parameter **chanNum**: the channel number from 0 ~ totalChans-1
Output Parameter **volume**: the retrieved volume for audio monitored on PC's speakers.
If audio monitoring is using WaveOutXX API(parameter **useWaveOutAPI** is true to function **MPEGIO2_audioMonitorStart**), the volume's low-order word contains the left-channel volume setting, and the high-order word contains the right-channel volume setting.
A value of 0xFFFF represents full volume, and a value of 0x0000 is silence.
If audio monitoring is not using WaveOutXX API(parameter **useWaveOutAPI** is false to function **MPEGIO2_audioMonitorStart**), thus using DirectSound API, the volume is specified in hundredths of decibels (dB):
possible values are between DSBVOLUME_MAX (0, no attenuation, maximum sound) and DSBVOLUME_MIN (-10000, silence).
Return true for success
Note: If the channel is not monitoring audio this function returns false

MPEGIO2_API bool **MPEGIO2_pauseAudioMonitor**(ULONG chanNum);

Function: Pause PC Audio Device if chanNum channel is currently previewing/monitoring audio
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return true for success
Note: This function is only applicable when audio preview/monitoring is using WaveOut API (paramter **useWaveOutAPI** to function **MPEGIO2_audioMonitorStart** is true).
If audio monitoring is not using WaveOut API then this function returns false.

MPEGIO2_API bool **MPEGIO2_resumeAudioMonitor**(ULONG chanNum);

Function: Re-Start Monitoring/Previewing PC Audio if chanNum channel is currently pausing audio monitoring by previously calling **MPEGIO2_pauseAudioMonitor**
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return true for success
Note: This function is only applicable when audio preview/monitoring is using WaveOut API (paramter **useWaveOutAPI** to function **MPEGIO2_audioMonitorStart** is true)
If audio monitoring is not using WaveOut API then this function returns false.

4.8 MPEG Encoding Functions

Encoding functions control how incoming video/audio is encoded into MPEG data: "encoding" means one or more of these operations:

Saving the MPEG data to data files (recording),
Streaming the data over IP network(streaming),
Passing the data to application-supplied callback functions (calling back).

MPEGIO2 SDK allows the application software to start/stop these encoding operations independently or simultaneously, on one or multiple **MPEGIO2** channels.

Note most of the encoding parameters can only be set when the channel is not in encoding mode.

4.8.1 Generic Encoding Functions

MPEGIO2_API bool **MPEGIO2_hasMPEGCodecIC**(unsigned long chanNum);

Function: Test if the MPEG Codec (Encoding/Decoding) IC is enabled for an **MPEGIO2** channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Return true if the **MPEGIO2** channel has its MPEG Encoding / Decoding Codec IC enabled, false if the MPEG Codec IC is disabled or faulty or driver is not installed

Note: When a channel's MPEG Codec IC is not enabled(e.g. through setting 1 in parameter "**NoCodecIC**" when calling function **MPEGIO2_initSDK**), that channel cannot use any MPEG encodig/recording/decoding/streaming function, although its video/audio preview functions can still work.

MPEGIO2_API bool **MPEGIO2_newEncoderStreamType**(unsigned long chanNum,
unsigned short newType);

Function: Change **MPEGIO2** channel's MPEG Encoder Stream Type
Input Parameter **chanNum**: **MPEGIO2** Channel Number, must be between 0~totalChans-1
Input Parameter **newType**: 0 for Program Stream (Default), 1 for Transport Stream
Return true for success.
Note 1: The channel must not be in encoding mode when this function is called or the function will fail.
Note 2: if the **newType** is the same as the channel's current Encoder type this function does nothing and returns true.
Note 3: A channel's MPEG **encoder stream type** decides how this channel encodes/decodes the MPEG data:

Program Stream encoder can only encode and decode Program Stream MPEG files/streams,
Transport Stream encoder can only encode and decode Transport Stream MPEG files/streams.

MPEGIO2_API bool **MPEGIO2_getEncoderStreamType**(unsigned long chanNum,
unsigned short &type);

Function: Return the channel's MPEG Encoder Type
Input Parameter **chanNum**: channel number, must be between 0~totalChans-1
Output parameter **type**: 0 for Program Stream, 1 for Transport Stream
Return: true for success.

MPEGIO2_API bool **MPEGIO2_startMPEGEncoding**(
unsigned long chanNum,
char *fileName,
bool startStreaming = false,
bool transcode = false,
unsigned short EncoderType = 0,
bool truncateFile = true);

Function: Start encoding MPEG on a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fileName**: recording file name including full path, must be valid file name, otherwise recording will not start(although streaming might start if parameter "**startStreaming**" is true)
Input parameter **startStreaming**: If true and the channel is not already streaming, will start streaming video datagram over IP network set up previously by calling **MPEGIO2_setupStreaming.**
If false, no effect on the streaming status of the channel.

Input parameter **transcode**: Currently ignored, must always be false

Input parameter **EncoderType**: MPEG_STREAM_TYPE_PS (0, Program Stream, default) or
MPEG_STREAM_TYPE_TS(1: Transport Stream)

Input parameter **truncateFile**: If true and the parameter "**fileName**" is an existing file name then the file is truncated to zero length before being written to,
If false and the parameter "**fileName**" is an existing file name then the file is appeneded to (encoded data is written to the end of the file)

Return true for success.

Note 1: A channel cannot start encoding while is decoding.

Note 2: If **filename** is not a valid file name, and **startStreaming** is false**,** then **MPEGIO2_setEncodeCBC** or **MPEGIO2_setEncodeCBS** must have been called to setup an application-supplied callback function so that this function can succeed in starting MPEG encoding: encoded data will be repeatedly passed to the application-supplied callback function. If **filename** is not a valid file name, and **startStreaming** is false and no application-supplied callback function is set up by calling **MPEGIO2_setEncodeCBC** or **MPEGIO2_setEncodeCBS**, then this function will fail and returns false.

```
MPEGIO2_API bool  MPEGIO2_stopMPEGEncoding(
                                unsigned long chanNum,
                                unsigned __int64 *encodeBytes = 0,
                                unsigned __int64 *streamBytes = 0,
                                unsigned __int64 *writeBytes = 0,
                                unsigned __int64 *writeBytesLastFile = 0,
                                unsigned long *splitFileNum = 0);
```

Function: Stop encoding MPEG data on a Channel.

Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number.

Output parameter **encodeBytes**: if not NULL must point to a variable to accept the encoded number of bytes.

Output parameter **streamBytes**: if not NULL must point to a variable to accept the streamed out number of bytes.

Output parameter **writeBytes**: if not NULL must point to a variable to accept the number of bytes written to all recorded files.

Output parameter **writeBytesLastFile**: if not NULL must point to a variable to accept the number of bytes written to the last recording file.

Output parameter **splitFileNum**: if not NULL must point to a variable to accept the number of split recording files.

Return true for success.

Note: On success, this function will stop recording, streaming and calling application-supplied callback function.

```
MPEGIO2_API bool  MPEGIO2_isMPEGEncoding(unsigned long chanNum);
```

Function: Return true if an **MPEGIO2** Channel is encoding

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.

```
MPEGIO2_API bool  MPEGIO2_pauseEncoding(unsigned long chanNum, bool pause);
```

Function: Pause or resume MPEG encoding on an **MPEGIO2** channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input parameter **pause**: true to pause encoding, false to resume encoding

Return: true for success

Note 1: this function disables the video signal input to MPEG encoding IC so even if encoding has not been started this function will still affect the MPEG encoding (pause or resume) when it's started subsequently

Note 2: pause/resume encoding will make MPEG file recording, streaming and calling callback functions paused or resumed

Note 3: if the channel is already pausing encoding passing parameter pause == true returns true without affecting anything, if the channel is still encoding passing parameter pause == false returns true without affecting anything.

```
MPEGIO2_API bool  MPEGIO2_isEncodingPaused(unsigned long chanNum, bool &paused);
```

Function: Return if encoding is paused on a channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Output parameter **paused**: true means encoding is paused, false: it is in progress or can start recording and/or streaming

Return: true for success

Note: this function merely tests if the MPEG encoding IC's encoding capability, it does not test if the user has started MPEG encoding by calling function **MPEGIO2_startMPEGEncoding, MPEGIO2_startRecording** or **MPEGIO2_startStreaming.** To test if a channel is encoding, call **MPEGIO2_isMPEGEncoding.**

MPEGIO2_API bool **MPEGIO2_getEncodeSizes**(
unsigned long chanNum,
unsigned int &recordTimeTotal,
unsigned int &recordTimeLastFile,
unsigned __int64 &accumulatedEncodeSize,
unsigned __int64 &accumulatedEncodeStreamSize,
unsigned __int64 &accumulatedWriteSize,
unsigned __int64 &lastFileWriteSize,
unsigned long &splitFileNum,
char *fileName = 0,
unsigned long fileNameLen = 0);

Function: Return the encoding sizes in bytes for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **recordTimeTotal**: the recording time (time to write to files) in seconds, inc. writing time to all split files
Output parameter **recordTimeLastFile**: the recording time for writing to the current file in seconds, time writing to previously split files are not counted
Output parameter **accumulatedEncodeSize**: bytes that have been encoded since encoding started
Output parameter **accumulatedEncodeStreamSize**: bytes that have been streamed out since streaming started
Output parameter **accumulatedWriteSize**: bytes that have been written to files since recording started
Output parameter **lastFileWriteSize**: bytes that have been written to the current file if recording is in progress
Output parameter **splitFileNum**: the current split file's number if split has ever been done, if no split has ever happened or no recording is in progress this is returned as 0
Output parameter **fileName**: if not NULL and **fileNameLen** is > 1, must point to a memory chunk at least fileNameLen bytes to hold the returned file name as the currently being recorded file name as a null-terminated string
Input parameter **fileNameLen**: if > 1, indicates the length in bytes of **fileName** parameter. If the current recording file name length is longer than this length the returned file name string is truncated to be equal to this value as a null-terminated string
Return true on success
Note: if the channel is not in encoding mode then values returned by this function might reflect previous recording result, or might be meaningless if no previous recording has ever happened.

MPEGIO2_API bool **MPEGIO2_newStreamType**(unsigned long chanNum, int st);
Function: Set new MPEG Encoding Stream Type (the stream type value encoded in the MPEG data) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **st**: for Program Stream Encoding, 1 <= **st** <=4, for Transport Stream Encoding, 0 <= **st** <=4:
For Program Stream Encoder:
1=VES, (Video Elementry Stream)
2=AES, (Audio Elementry Stream)
3=System Stream, (MPEG1)
4=Program Stream(Default)
For Transport Stream Encoder:
0=Transport Stream(Default),
1=Fixed pattern (test mode),
2=AES, (Audio Elementry Stream)
3=VES, (Video Elementry Stream)
4=Video element stream with BCH coding
Return true for success
Note 1: The channel must not be in encoding mode when this function is called or the function will fail.
Note 2: Currently Transport Stream Encoder Can Only Support **st**==0, AES/VES are not available for TS Encoder Encoding!
Note 3: AES result currently is not playable so do not use it!

MPEGIO2_API bool **MPEGIO2_getStreamType**(unsigned long chanNum, int &st);
Function: Get the MPEG Encoding Stream Type (the stream type value encoded in the MPEG data) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **st**: MPEG encoding stream type setting, see **MPEGIO2_newStreamType** for their explanations.
Return true for success.

MPEGIO2_API bool **MPEGIO2_newEncodingType**(unsigned long chanNum, int type);
Function: Set MPEG encoding type on a channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **type** 1: MPEG1, 2: MPEG2, 4: MPEG4
Return true success.
Note 1: the channel must not be in encoding mode when this function is called or the function will fail.
Note 2: If **type** is 1 (MPEG1), the encoding HSize (see **MPEGIO2_newEncodingHSize**)must be <= 352, and the encoding VSize (see **MPEGIO2_newEncodingVSize**)must be <= 288 for PAL, and <= 240 for NTSC.
Note 3: The MPEG2/MPEG1 encoded video can be played by most hardware/software players, but MPEG4 video playback is less universal: known software that can play **MPEGIO2** encoded MPEG4 video include VideoLan(vlc.exe), SMPlayer(MPlayer), MainConcept ShowCase, ffplay.exe(ffmpeg), DiVX Plus Player, Total Video Player, Elecard MPEG Player, etc. MPEG4 encodes higher quality at low bit rates when compared with MPEG2 encoding at the same low bit rates.

MPEGIO2_API int **MPEGIO2_getEncodingType**(unsigned long chanNum);
Function: Get MPEG encoding type on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 1: MPEG1, 2: MPEG2, 4: MPEG4 , 0 : failure.

MPEGIO2_API bool **MPEGIO2_newInitialTimeCode**(unsigned long chanNum,
int hour,
int min,
int sec,
int frame,
int drop_frame_flag);
Function: Set the initial SMPTE time code for the first encoded video frame for a channel when it next starts encoding
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **hour**: 0~23, the hour value of the time code
Input parameter **min**: 0~59, the minute value of the time code
Input parameter **sec**: 0~59, the seconds value of the time code
Input parameter **frame**: the picture number of the time code
Input parameter **drop_frame_flag**: this is used to support 29.97fps in NTSC mode.
If set to 1, then picture numbers 0 and 1 at the start of each minute (except minute 0, 10, 20, 30, 40, 50) are omitted from the count. For PAL mode, this flag is ignored.
Return true success
Note 1: This function can only be called before the channel starts encoding (inc. recording and streaming or doing both). Calling it when encoding or decoding is in progress will get false return value and nothing will be set.
Note 2: default time code for encoding is 0.
Note 3: This function work for either Program Stream or Transport Stream Encoder Type: but they need to be set separately.

MPEGIO2_API bool **MPEGIO2_getInitialTimeCode**(unsigned long chanNum,
int &hour,
int &min,
int &sec,
int &frame,
int &drop_frame_flag);
Function: Get the initial SMPTE time code used for the first encoded video frame for a channel when it encodes.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Output parameter **hour**: 0~23, the hour value of the time code
Output parameter **min**: 0~59, the minute value of the time code
Output parameter **sec**: 0~59, the seconds value of the time code
Output parameter **frame**: the picture number of the time code
Output parameter **drop_frame_flag**: this is used to support 29.97fps in NTSC mode.
See explanation in function "**MPEGIO2_getInitialTimeCode**".
Return true success.

MPEGIO2_API bool **MPEGIO2_newVOBFormat**(unsigned long chanNum, int vob);
Function: Set VOB encoding format for an **MPEGIO2** channel that is configured as Program Stream Encoder Type
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **vob**: 1(Default)=Set the Program Stream Encoding output format to VOB,
0=Not using VOB format(use standard PS format)
Return true success.

Note: This function is valid only for Program Stream encoder and the channel is not currently encoding.

MPEGIO2_API bool **MPEGIO2_getVOBFormat**(unsigned long chanNum, int &vob);
Function: Get VOB encoding format for a channel that is configured asProgram Stream Encoder Type
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **vob**: 1= VOB format,
0= Not VOB format(standard PS format)
Return true success.
Note: This function is valid only for Program Stream encoder, if the channel is Transport Stream Encoder it returns false.

4.8.2 Video Encoding Functions

MPEGIO2_API bool **MPEGIO2_newMPEGVideoStandard**(unsigned long chanNum, unsigned long videoStandard);
Function: Set MPEG encoding/decoding video standard for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **videoStandard**: NTSC=1, PAL=2
Return true for success
Note 1: Before start setting other encoding parameters such as horizontal/vertical frame sizes etc., this function should be called, otherwise some frame size combinations might not be successfully set
Note 2: A Channel cannot set video standard while is encoding or decoding.

MPEGIO2_API bool **MPEGIO2_newEncodingBitRate**(unsigned long chanNum, int bitrate);
Function: Set MPEG Encoding Bit rate on a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **bitrate**: Must be within 132K ~ 15000 K(1024)-bits per second(bps): must be approximately limited according to encoding video's Horizontal Frame Size (see function **MPEGIO2_newEncodingHSize**):
1573 Kbps to 15 Mbps if horizontal frame size is > 352-Pixel; (Note 1 Mbps = 1024 Kbps)
525 Kbps to 15 Mbps if horizontal frame size is 352-Pixel when PAL, frame size is 320-Pixel when NTSC;
132 Kbps to 15 Mbps if horizontal frame size <= 176-Pixel.
Return true for success.
Note: the channel must not be encoding when this function is called or the function will fail.

MPEGIO2_API int **MPEGIO2_getEncodingBitRate**(unsigned long chanNum);
Function: Return MPEG Encoding Bit rate on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise the current encoding bit rate as K-bits per second unit.

MPEGIO2_API bool **MPEGIO2_newEncodingBitRateAverage**(unsigned long chanNum, int bitrate);
Function: Set MPEG Average Encoding Bit rate on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **bitrate**: Must be within 132K ~ 15000K-bits per second and <= **MPEGIO2_getEncodingBitRate.**
Return true for success.
Note: the channel must not be CRB and not in encoding mode when this function is called or the function will fail

MPEGIO2_API int **MPEGIO2_getEncodingBitRateAverage**(unsigned long chanNum);
Function: Get MPEG Average Encoding Bit rate on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise the current average encoding bit rate as K-bits per second unit.

MPEGIO2_API bool **MPEGIO2_newEncodingCBR**(unsigned long chanNum, bool cbr);
Function: Set MPEG Encoding CBR (or VBR) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **cbr**: true for CBR(Constant Bit Rate), false for VBR(Variable Bit Rate)
Return true for success.
Note: the channel must not be in encoding mode when this function is called or the function will fail

MPEGIO2_API bool **MPEGIO2_isEncodingCBR**(unsigned long chanNum);
Function: Return if MPEG Encoding is CBR or VBR on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Return true for CBR, false for VBR.

MPEGIO2_API bool **MPEGIO2_newEncodingVSize**(unsigned long chanNum, int size);
Function: Set MPEG encoding video frame vertical size in pixels on a channel when it is not currently encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **size**: for PAL: 144 , 288, 576, NTSC: 144, 240, 288, 480, 512
Return true success.
Note 1: If encoding MPEG Type (See **MPEGIO2_newEncodingType**) is 1 (MPEG1),
then **size** must be <= 288 for PAL encoding, **size** must be <= 240 for NTSC encoding.
Note 2: For NTSC encoding and all MPEG1,2,4 types, when Horizontal Frame Size is 352, Vertical Size must be <= 240.

MPEGIO2_API int **MPEGIO2_getEncodingVSize**(unsigned long chanNum);
Function: Get MPEG encoding video frame vertical size in pixels on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise the MPEG encoding video frame vertical size in pixels.

MPEGIO2_API bool **MPEGIO2_newEncodingHSize**(unsigned long chanNum, int size);
Function: Set MPEG encoding video frame horizontal size in pixels on a channel when it is not currently encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **size**: one of 176, 320, 352, 360, 480, 528, 544, 640, 704, 720
Return true success.
Note: If encoding MPEG Type (See **MPEGIO2_newEncodingType**) is 1 (MPEG1), then **size** must be <= 352.

MPEGIO2_API bool **MPEGIO2_getEncodingHSize**(unsigned long chanNum);
Function: Get MPEG encoding video frame horizontal size in pixels on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise the MPEG encoding video frame horizontal size in pixels.

MPEGIO2_API bool **MPEGIO2_newHSP**(unsigned long chanNum, int hsp);
Function: Set new Horizontal Start Position inside video frame in encording on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **hsp**: pixel number to start MPEG encoding horizontally, default is -1(firmware auto determines this).
Return true success.

MPEGIO2_API bool **MPEGIO2_getHSP**(unsigned long chanNum, int &hsp);
Function: Get Horizontal Start Position inside video frame in encording on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **hsp**: pixel number to start MPEG encoding horizontally, -1 means the firmware auto determines this.
Return true success.

MPEGIO2_API bool **MPEGIO2_newVSP**(unsigned long chanNum, int vsp);
Function: Set new Vertical Start Position inside video frame in encording on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **vsp**: pixel line number to start MPEG encoding vertically,
default is -1(start from 0 for both odd and even field).
Return true success.

MPEGIO2_API bool **MPEGIO2_getVSP**(unsigned long chanNum, int &vsp);
Function: Get Vertical Start Position inside video frame in encording on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **vsp**: pixel line number to start MPEG encoding vertically,
-1 means start from 0 for both odd and even field.
Return true success.

MPEGIO2_API bool **MPEGIO2_newEncodingAR**(unsigned long chanNum, int ar);
MPEGIO2_API bool **MPEGIO2_encoderSet_aspectRatio**(unsigned long chanNum, int ar);
Function: Set new encoding aspect ratio on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **ar**: 1=Square PEL(1:1), 2(default)=4:3, 3=16:9, 4=2.21:1
Return true success.

MPEGIO2_API int **MPEGIO2_getEncodingAR**(unsigned long chanNum);
Function: Get new encoding aspect ratio on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise aspect ratio: 1=Square PEL(1:1), 2=4:3, 3=16:9, 4=2.21:1.

MPEGIO2_API bool **MPEGIO2_encoderGet_aspectRatio**(unsigned long chanNum, int &ar);
Function: Get current encoding aspect ratio on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **ar**: 1=Square PEL, 2(default)=4:3, 3=16:9, 4=2.21:1
Return true success.

MPEGIO2_API bool **MPEGIO2_newEncodingFrameRate**(unsigned long chanNum, int newfr);
Function: Set MPEG encoding frame rate on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **newfr**: Frame Rate in frame per second (fps):
1=23.976,
2=24,
3=25(default for PAL),
4=29.97(default for NTSC),
5=30,
6=50,
7:59.94,
8:60.
Return true for success.

MPEGIO2_API bool **MPEGIO2_getEncodingFrameRate**(unsigned long chanNum, int &fr);
Function: Get MPEG encoding frame rate on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **fr**: Frame Rate in frame per second (fps):
Return true for success.

MPEGIO2_API bool **MPEGIO2_newProgressiveSequence**(unsigned long chanNum,
int isProgressive);
Function: Set MPEG encoding as progressive or interleaved sequence on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **isProgressive**: 1 for progressive, 0 for interleaved (default)
Return true for success.

MPEGIO2_API bool **MPEGIO2_getProgressiveSequence**(unsigned long chanNum,
int &isProgressive);
Function: Get MPEG encoding progressive or interleaved sequence setting on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **isProgressive**: 1 for progressive, 0 for interleaved (default)
Return true for success.

MPEGIO2_API bool **MPEGIO2_new_mpeg4_quant_type**(unsigned long chanNum, int type);
Function: Set the quantization type for MPEG-4 encoding on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **type**: 0(H.263) or 1(MPEG-2)=Default
Return true for success.

MPEGIO2_API bool **MPEGIO2_get_mpeg4_quant_type**(unsigned long chanNum, int &type);
Function: Get the quantization type for MPEG-4 encoding on a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **type**: receives 0(H.263) or 1(MPEG-2)
Return true for success.

MPEGIO2_API bool **MPEGIO2_new_mpeg4_short_header** (unsigned long chanNum, int header);
Function: Set MPEG-4 short header and H.263 mode when encoding in MPEG4 Format for a channel that is not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input parameter **header**: 0 -- MPEG-4 only (Default),
1 -- MPEG-4 short header (allows encapsulation of H.263 bit stream),
2 -- H.263.
Return true for success.

MPEGIO2_API bool **MPEGIO2_get_mpeg4_short_header** (unsigned long chanNum, int &header);
Function: Get MPEG-4 short header and H.263 mode when encoding in MPEG4 Format for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **header**: Same as in function **MPEGIO2_new_mpeg4_short_header**
Return true for success.

MPEGIO2_API bool **MPEGIO2_newBlackScreen**(unsigned long chanNum, int black);
Function: Set MPEG encoding as black screen(no video content) on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input parameter **black**: 1 for set as black screen, 0 for not (default) black screen (normal video)
Return true for success.

MPEGIO2_API bool **MPEGIO2_getBlackScreen**(unsigned long chanNum, int &black);
Function: Get MPEG encoding's black screen(no video content) setting on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **black**: 1 for set as black screen, 0 for not (default) black screen (normal video)
Return true for success.

MPEGIO2_API bool **MPEGIO2_newSceneChangeDetection**(unsigned long chanNum, int sc);
Function: Set the scene change detection in the video encoder on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **sc**: 1 for scene change detect, 0 for not detect(default) scene change
Return true for success.

MPEGIO2_API bool **MPEGIO2_getSceneChangeDetection**(unsigned long chanNum, &int sc);
Function: Get the scene change detection setting in the video encoder on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **sc**: 1 for scene change detect, 0(default) for not detect scene change
Return true for success.

MPEGIO2_API bool **MPEGIO2_updateVideoParam**(unsigned long chanNum, int update);
Function: Update the encoding parameters set prior to this call during encoding on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **update**: 1 = do the udates, 0 (default) do not do the updates
Return true for success
Note: Some encoding parameters such as user data, can be dynamically updated during encoding by first call the relevant functions to change the encoding parameters then call this function to update their values dynamically.

4.8.3 Audio Encoding Functions

MPEGIO2_API bool **MPEGIO2_newEncodingAudioParameters**(
unsigned long chanNum,
int format,
int samplingRate,
int bitRate);

Function: Set MPEG encoding audio parameters on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **format**: Audio encoding format:
0:AC3 or MPEG1 Layer 1,
1:MPEG1 Layer 2,
2:MPEG1 Layer 3.
Input parameter **samplingRate**: Audio encoding sampling rate: 1:32KHz, 2: 44.1KHz, 3: 48KHz:
the encoding audio sampling rate set here must match the Audio Preview Sampling Rate set by function **MPEGIO2_setAudioStandard**.
Input parameter **bitRate**: Audio encoding bit rate in bits per second:
For Program Stream Encoder:
32000, 48000, 56000, 64000, 80000, 96000, 112000, 128000,
160000, 192000, 224000, 256000, 320000, 384000
For Transport Stream Encoder: if format == 0:
32000, 64000, 96000, 128000, 160000, 192000, 224000,
256000, 288000, 320000, 352000, 384000, 416000, 448000
For Transport Stream: if format != 0:
32000, 48000, 56000, 64000, 80000, 96000, 112000, 128000,
160000, 192000, 224000, 256000, 320000, 384000.
Return true success
Note 1: This function must be called when the channel is NOT encoding or failure will occure.
Note 2: Format G72X are mono encoding with audio signal taken from the left audio sub-channel.

MPEGIO2_API bool **MPEGIO2_getEncodingAudioParameters**(
unsigned long chanNum,
int &format,
int &samplingRate,
int &bitRate);

Function: Get current MPEG encoding audio parameters on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **format**: Audio encoding format: Same as in **MPEGIO2_newEncodingAudioParameters:**
0:AC3/MPEG1L1, 1:MPEG1L2, 2:MPEG1L3.
Output parameter **samplingRate**: Same as in **MPEGIO2_newEncodingAudioParameters**
Output parameter **bitRate**: Same as in **MPEGIO2_newEncodingAudioParameters**
Return true success.

MPEGIO2_API bool **MPEGIO2_newEncodingAudioLayer**(unsigned long chanNum, int al);
Function: Set MPEG encoding audio layer on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **al**: For Program Stream: 1:MPEG1L2, 2:MPEG1L2(Default), 3:MPEG1L3(MP3).
For Transport Stream: 1 -- MPEG-1 Layer 1
2 -- MPEG-1 Layer 2(Default)
3 -- MPEG-1 Layer 3 (MP3)
0x81 -- AC3
Return true for success
Note: This function must be called when the channel is NOT encoding or failure will occure.

MPEGIO2_API bool **MPEGIO2_getEncodingAudioLayer**(unsigned long chanNum, int &al);
Function: Get MPEG encoding audio layer on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **al**: For Program Stream: 1:MPEG1L2, 2:MPEG1L2, 3:MPEG1L3(MP3).
For Transport Stream: 1 -- MPEG-1 Layer 1

2 -- MPEG-1 Layer 2
3 -- MPEG-1 Layer 3 (MP3)
0x81 -- AC3

Return true for success.

MPEGIO2_API bool **MPEGIO2_newEncodingAudioStreamType**(unsigned long chanNum, int ast);

Function: Set MPEG encoding audio stream type (Program Stream Only) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **ast**: Audio Stream Type Value =
1 -- MPEG-1 Audio (Default)
2 -- MPEG-2 Audio
6 -- AC3
Return true for success.
Note 1: This function must be called when the channel is NOT encoding or failure will occure!
Note 2: This function is only applicable when the channel is Program Stream Encoder.

MPEGIO2_API bool **MPEGIO2_getEncodingAudioStreamType**(unsigned long chanNum, int &ast);

Function: Get MPEG encoding audio stream type (Program Stream Only) on a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **ast**: Audio Stream Type Value =
1 -- MPEG-1 Audio (Default)
2 -- MPEG-2 Audio
6 -- AC3
Return true for success
Note: This function is only applicable when the channel is Program Stream Encoder.

MPEGIO2_API bool **MPEGIO2_newEncodingAudioSamplingRate**(
unsigned long chanNum,
int asr);

Function: Set new MPEG encoding audio sampling rate on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **asr**: Audio Sampling Rate: must be one of 32000/44100/48000 Hz.
Return true for success.

MPEGIO2_API int **MPEGIO2_getEncodingAudioSamplingRate**(
unsigned long chanNum);

Function: Get the current MPEG encoding audio sampling rate on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return 0 for failure, otherwise return the "**Audio Sampling Rate**" as described in parameter "**asr**" of function "**MPEGIO2_newEncodingAudioSamplingRate**".

MPEGIO2_API bool **MPEGIO2_newAudioChannelMode**(unsigned long chanNum, int acm);

Function: Set new MPEG encoding audio channel coding mode on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **acm**: must be 0 ~ 3 : 0(default)=stereo, 1=joint stereo, 2=dual channel, 3=single channel
Return true for success.

MPEGIO2_API bool **MPEGIO2_getAudioChannelMode**(unsigned long chanNum, int &acm);

Function: Get MPEG encoding audio channel coding mode on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **acm**: can be 0 ~ 3 : 0(default)=stereo, 1=joint stereo, 2=dual channel, 3=single channel
Return true for success.

4.8.4 GOP Structure Functions

MPEGIO2_API bool **MPEGIO2_newEncodingGOP**(unsigned long chanNum, int N, int M);
Function: Set new encoding GOP(Groupf of Picture) structure N & M values on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **N**: GOP Parameter N (1~256)
Input parameter **M**: GOP Parameter M (0,1,2,3)
Return true success.
Note:

If M is not 0, then N must be a multiple of M.
If M = 0, then N must be set to 1.
This function set the group_of_picture (GOP) structure via the parameters N (number of frames in GOP) and M (frame distance between reference frames).
6 <= N < 256 in integer multiples of M for M = 3
4 <= N < 256 in integer multiples of M for M = 2
2 <= N < 256 in integer multiples of M for M = 1
For example,
If N=15 and M=3, the GOP structure is I B B P B B P B B P B B P B B.
If M = 0, the GOP structure is I (encoder will generate I frames only)
If M = 1, the GOP structure is IP
If M = 2, the GOP structure is IBP
If M = 3, the GOP structure is IBBP
See ISO/IEC 13818-2 sections 6.2.2.6 and 6.3.8 "Group of pictures header."
Default Values N=15 and M=3.

MPEGIO2_API bool **MPEGIO2_getEncodingGOP**(unsigned long chanNum, int &N, int &M);
Function: Get current encoding GOP structure N & M values on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **N**: GOP Parameter N (1~256)
Output parameter **M**: GOP Parameter M (0,1,2,3)
Return true success.

MPEGIO2_API bool **MPEGIO2_newClosedGOP**(unsigned long chanNum, int cgop);
Function: Set clopsed GOP for encoding on a channel when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **cgop**: must be 1 (closed GOP) or 0 (not closed GOP, default)
Return true for success.

MPEGIO2_API bool **MPEGIO2_getClosedGOP**(unsigned long chanNum, int &cgop);
Function: Get the clopsed GOP setup for encoding on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **cgop**: 1 (closed GOP) or 0 (not closed GOP, default)
Return true for success.

4.8.5 PID Functions

MPEGIO2_API bool **MPEGIO2_encoderSet_video_pid**(unsigned long chanNum, int pid);
Function: Set the Video Program ID (PID) for a channel opened as a Transport Stream Encoder when it's not encoding.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pid**: the new video PID, must be within 21 ~ 8190, default is 0x21(33)
Return true success.
Note: If the channel is not a Transport Stream encoder or if the channel is encoding this function returns false.

MPEGIO2_API bool **MPEGIO2_encoderSet_audio1_pid**(unsigned long chanNum, int pid);
Function: Set the Audio 1 (first audio stream) Program ID (PID) for a channel opened as a Transport Stream Encoder
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pid**: the new Audio 1 PID, must be within 21 ~ 8190, default is 0x22(34)
Return true success
Note: If the channel is not a Transport Stream encoder, or if the channel is encoding, this function returns false.

MPEGIO2_API bool **MPEGIO2_encoderSet_audio2_pid**(unsigned long chanNum, int pid);
Function: Set the Audio 2 (second audio stream) Program ID (PID) for a channel opened as a Transport Stream Encoder
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pid**: the new Audio 2 PID, must be 0 or 1, default is 0
Return true success
Note: If the channel is not a Transport Stream encoder or if the channel is encoding this function returns false.

MPEGIO2_API bool **MPEGIO2_encoderSet_pcr_pid**(unsigned long chanNum, int pid);
Function: Set the program clock reference (PCR) PID (a 13-bit field) for the specified program number for an **MPEGIO2** channel configured as a Transport Stream Encoder
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pid**: the new PCR PID value, must be within 21(0x15) ~ 8190(0x1FFE), default is 0x25
Return true success
Note: If the channel is not a Transport Stream encoder or if the channel is encoding this function returns false.

MPEGIO2_API bool **MPEGIO2_encoderGetPIDs**(unsigned long chanNum,
int &video_pid,
int &audio1_pid,
int &audio2_pid,
int &pcr_pid,
int &user_data_pid);
Function: Get various PIDs for an **MPEGIO2** channel configured as a Transport Stream Encoder
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **video_pid**: the current video PID
Output parameter **audio1_pid**: the current audio 1 PID
Output parameter **audio2_pid**: the current audio 2 PID
Output parameter **pcr_pid**: the current program clock reference PID
Output parameter **user_data_pid**: the user data (insertion) PID
Return true success
Note: If the channel is not a Transport Stream encoder this function returns false

4.8.6 User Data Insertion Functions

MPEGIO2_API bool **MPEGIO2_insertUserData**(unsigned long chanNum,
int user_data_length,
char *user_data);

Function: Insert user data bytes into a channel if it is currently encoding or when it next time starts encoding
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **user_data_length**: The bytes of data contained in "user_data" array,
must be within 1~120 if this channel is Program Stream Encoder
or within 1 ~ 176 if this channel is Transport Stream Encoder.
Input parameter **user_data**: must point to a byte array of at least **user_data_length** long, the content of this parameter will be inserted into the current MPEG encoding stream if this channel is currently encoding, or they will be inserted into the MPEG encoding stream when next time the channel starts encoding.
Return true for success.
Note 1: When the "**user_data_length**" bytes data are inserted the channel will wait until next call to this function to insert new or same user data
Note 2: The actual data inserted into the MPEG stream will be 3 NULL bytes (0x00) followed by a one byte indicating user_data_length value then followed by the "**user_data**" content supplied here.
Note 3: The frequency of the user data insertion should not be higher than the video frame rate.
Note 4: When the **MPEGIO2** channel is configured as Transport Stream Encoder, if the user data PID (see "**id**" in **MPEGIO2_newTSUserDataPID**) and the video PID (see "**VPID**" in **MPEGIO2_startMPEGEncoding**) are the same, the **MPEGIO2** will add the user data to the adaptation_field of the video packet; if the user data PID and the video PID are different, the data packet will contain user data.
Note 5: This function will automatically call **MPEGIO2_updateVideoParam** after sending the user data to the encoding **MPEGIO2** channel.
Note 6: If **MPEGIO2_notifyUserDataInsertion**(chanNum, true) was previously called, a MPEGIO2_MSG_USER_DATA_INSERT message will be sent back from **MPEGIO2** SDK to the application window indicated by parameter "**parentWnd**" passed to the **MPEGIO2_initSDK** function.
Note 7: Inserted user data is not remembered by the **MPEGIO2** SDK: once inserted they will be discarded immediately.

MPEGIO2_API bool **MPEGIO2_notifyUserDataInsertion**(unsigned long chanNum, bool notify);
Function: Enable/disable sending notify message on user data insertion for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **notify**: if true, an MPEGIO2_MSG_USER_DATA_INSERT message will be sent to application window as indicated by the parameter "**parentWnd**" passed to the **MPEGIO2_initSDK** function. If false, no message will be sent.

MPEGIO2_API bool **MPEGIO2_newTSUserDataPID**(unsigned long chanNum, int id);
Function: Set Transport Stream Encoder's User Data Program ID for an **MPEGIO2** channel
(the program identification -- PID, a 13-bit field -- for the user data stream specified in the TS program map section)
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **id**: The new TS Stream User Data Program ID (default is 36 or 0x24), valid within 1 ~ 183 inclusive.
Return true for success
Note: The channel must be configured as Transport Stream Encoder(see **MPEGIO2_newEncoderStreamType** and **MPEGIO2_startMPEGEncoding**) to make this fuction call to succeed.

4.9 MPEG Recording Functions

Recording means saving encoded MPEG data to files. A channel in recording mode must also be encoding, but a channel in encoding mode might not necessarily be recording since it might be simply streaming or calling application-supplied callback functions for its encoded MPEG data.
When a channel is recording, it can split the recording file: close the current file and open a new file then write all subsequent MPEG data to this new file. Files split in this way can later be stitched together to form a file containing no gaps at the time between every two ajacent split files since the splitting is "seamless".

MPEGIO2_API bool **MPEGIO2_startRecording**(unsigned long chanNum, char *fileName);
Function: Start file recording for an **MPEGIO2** Channel if it is already encoding but not recording
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fileName**: recording file name including full path, must be valid file name
Returns true if the channel started recording, otherwise returns false
Note: if the channel is not in encoding mode or is already in recording mode then this function returns false.

MPEGIO2_API bool **MPEGIO2_stopRecording**(unsigned long chanNum,
unsigned __int64 *writeBytes = 0,
unsigned __int64 *writeBytesLastFile = 0,
unsigned long *splitFileNum = 0);
Function: Stop file recording for a channel if it is recording
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **writeBytes**: if not NULL will receive total bytes written to all files
Output parameter **writeBytesLastFile**: if not NULL will receive the bytes written to the last recording file
Output parameter **splitFileNum**: if not NULL must point to a variable to accept the number of split files
Returns true for success.
Note 1: If the channel is not in encoding mode or not recording then this function returns false.
Note 2: If the channel is recording and streaming this function stops recording but the streaming continues therefore the channel is stil encoding(with streaming but not recording to file any more),
If the channel is only recording(writing to file) but not streaming this function stops the encoding entirely.
If the channel is calling application-supplied callback functions the callback will continue after calling this function.

MPEGIO2_API bool **MPEGIO2_isRecording**(unsigned long chanNum);
Function: Return true if an **MPEGIO2** Channel is recording(write encoded data to file)
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Returns true if the channel is recording, otherwise returns false
Note 1: if the channel is not in encoding mode then this function returns false.
Note 2: if the channel is paused in recording (it has a recording file opened but not writing data to it) this function still returns true.

MPEGIO2_API bool **MPEGIO2_splitRecording**(unsigned long chanNum);
Function: Split recording(stop current recording file, create a new recording file) for a channel that is currently recording
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Return true on success
Note 1: If the channel is not in recording mode then this function returns false
Note 2: Each split file name will be the very first recording file name plus an 8-digit numerical number appended at the end plus the ".mpg" extention:
the 8-digit numerical number starts from 00000001 then increases 1 for each new split file (00000002, 00000003,...).
If the **repeatSplitFileNum** is set to non-zero by function **MPEGIO2_setupSplitRecord**(), then when the **split file number** reaches **repeatSplitFileNum** the next split file will use the very first file name again without appending any numerical number, the next split file after this will use that file name with 00000001 appended to its end plus ".mpg", etc, ...

MPEGIO2_API bool **MPEGIO2_setupSplitRecord**(unsigned long chanNum,
unsigned long splitFileSize,
unsigned long splitFileTime,
unsigned long repeatSplitFileNum);
Function: Setup split recording parameters for a Channel

Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input parameter **splitFileSize**: if not 0, set the new file size in KBytes for splitting: when current recording file size reaches this value a new file will be split
Input parameter **splitFileTime**: if not 0, set the new time in minutes for splitting: when current recording time reaches this value a new file will be split
Input parameter **repeatSplitFileNum**: if not 0, and parameter splitFileSize or splitFileTime is also not 0, when file split this many times the split number appended to the original file name will reset to NULL (the very first recording file after this reset will have no serial number appended to its file name, the 1st split file after this reset will have 00000001 appeneded to its file name, ...)
Return true on success
Note: If both **splitFileSize** and **splitFileTime** are non-zero**,** recording file split could happen when either recording file size reaches **splitFileSize** or recording time reaches **splitFileTime.**

MPEGIO2_API bool **MPEGIO2_getSplitRecord**(unsigned long chanNum,
unsigned long &splitFileSize,
unsigned long &splitFileTime,
unsigned long &splitFileNum,
unsigned long &repeatSplitFileNum);

Function: Get the split recording parameters for an **MPEGIO2** Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **splitFileSize**: if not 0, is the file size in KBytes for splitting: when current recording file size reaches this value a new file will be split
Output parameter **splitFileTime**: if not 0, is the time in minutes for splitting: when current recording time reaches this value a new file will be split
Output parameter **splitFileNum**: the numerical number to append to the file name when next split happens
Output parameter **repeatSplitFileNum**: if not 0, and parameter **splitFileSize** or **splitFileTime** is also not 0, when file split this many times the split number appended to the original file name will reset to NULL (the very first recording file after this reset will have no serial number appended to its file name, the 1st split file after this reset will have 00000001 appeneded to its file name, ...)
Return true on success.

MPEGIO2_API bool **MPEGIO2_pauseRecording**(unsigned long chanNum, bool pause);

Function: Pause or resume MPEG recording(file writing) on an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pause**: true to pause recording, false to resume recording
Return: true for success
Note 1: this function only affects a channel's file writing during its encoding: streaming is not paused
Note 2: if the channel is not recording this function returns false
Note 3: if the channel is already pausing recording passing parameter pause == true returns true without affecting anything, if the channel is still recording passing parameter pause == false returns true without affecting anything.

MPEGIO2_API bool **MPEGIO2_isRecordingPaused**(unsigned long chanNum);

Function: Test is file recording is being paused as a result of calling **MPEGIO2_pauseEncoding**(chanNum, true) or calling **MPEGIO2_pauseRecording**(chanNum, true) for a channel when it is in recording mode.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Returns true if the channel is being paused recording
Note: if the channel is not in recording mode this function returns false.

MPEGIO2_API bool **MPEGIO2_setRecordingTimer**(unsigned long chanNum,
unsigned long minutes);

Function: Set file recording timer value for a channel regardless currently it is recording or not
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **minutes**: recording timer in munites, 0(default) means no timer (recording only ends manually)
Returns true for success.
Note: Once set, this timer will be automatically remembered by the SDK for this channel even between SDK exit/restart (through **initFile** MPEGIO2.ini)

MPEGIO2_API unsigned long **MPEGIO2_getRecordingTimer**(unsigned long chanNum);

Function: Get file recording timer value for a channel, regardless it is currently recording or not.

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Returns recording timer value. 0 means no timer or failure.

```
MPEGIO2_API bool  MPEGIO2_setFileNames(  unsigned long chanNum,
                                         char *recFile,
                                         char *imageFile);
```

Function: Set video recording and still image capturing file names on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **recFile**: if not NULL, must point to the new recording file name inc. path and terminated by NULL
Input parameter **imageFile**: if not NULL, must point to the new image file name inc. path and terminated by NULL
Return: true for success.
Note: Caller must ensure the **recFile** and **imageFile** are valid file names not contaning illegal characters such as ',', ':' etc.

```
MPEGIO2_API bool  MPEGIO2_getFileNames(  unsigned long chanNum,
                                         char **recFile,
                                         char **imageFile);
```

Function: Get video recording and still image capturing file and path names on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **recFile**: if not NULL, will point to the channel's latest used or defined recording file name inc. full path.
Output parameter **imageFile**: if not NULL, will point to the channel's still image capturing file name inc. full path.
Return: true for success.

4.10 MPEG Streaming Functions

MPEGIO2 SDK can stream out UDP datagram MPEG data for each channel during its encoding or decoding operation. Streaming can start/stop independent of recording or decoding.

MPEGIO2_API bool **MPEGIO2_resetNetwork**(void);
Function: Re-initialize the MS Windows' Networking
Return true for success
Note: if any channel is currently streaming this function will return false and does not initialize the Windows' networking.

```
MPEGIO2_API bool MPEGIO2_setupStreaming(unsigned long chanNum,
                                        char *addr,
                                        int port,
                                        bool multicast,
                                        int sendBufSize = 0,
                                        int recvBufSize = 0);
```

Function: Prepare streaming parameters for an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **addr**: a string in the form of XXX.XXX.XXX.XXX representing a valid IP address
Input parameter **port**: an TCP/IP port number
Input parameter **multicast**: true to setup a multicast streaming socket, false to set up a unicast socekt for streaming
Input parameter **sendBufSize**: the socket's send buffer size, if 0 is supplied (defaut), 128K will be used
Input parameter **recvBufSize**: the socket's receive buffer size, if 0 is supplied (defaut), 128K will be used
Return true on success, failure conditions include the PC is not ready for network operation etc.

MPEGIO2_API bool **MPEGIO2_startStreaming**(unsigned long chanNum);
Function: Start streaming if a channel is not streaming
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return true on success
Note 1: This function will set the channel to start streaming MPEG data if it is in either encoding or decoding mode
Note 2: If the channel is not encoding nor decoding this function will start MPEG encoding and Streaming simultaneously
Note 3: If the channel is decoding and is using network stream input as the decoding source this function returns false
Note 4: If the channel is already in streaming (during encoding or decoding) this function returns true and does nothing.

```
MPEGIO2_API bool MPEGIO2_getStreamingParameters(
                                        unsigned long chanNum,
                                        char *addr,
                                        int &port,
                                        bool &multicast,
                                        bool &isStreaming,
                                        int &sendBufSize,
                                        int &recvBufSize);
```

Function: Return the current streaming parameters for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **addr**: if not NULL, must be a string at least 15 characters long to hold a valid IP address in the form of XXX.XXX.XXX.XXX
Output parameter **port**: the TCP/IP port number
Output parameter **multicast**: true is using multicast streaming socket, false is using unicast socekt for streaming
Output parameter **isStreaming**: if true the channel is streaming out MPEG data to network
Output parameter **sendBufSize**: the socket's send buffer size, if 0 (defaut), 128K will be used
Output parameter **recvBufSize**: the socket's receive buffer size, if 0 defaut), 128K will be used
Return true on success.

MPEGIO2_API bool **MPEGIO2_isStreaming**(unsigned long chanNum);
Function: returns true if a channel is streaming (if yes then it must also be encoding or decoding)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.

MPEGIO2_API bool **MPEGIO2_isStreamingAvailable**(void);
Function: Returns true if the SDK on the PC can stream through TCP/IP network.

MPEGIO2_API bool **MPEGIO2_stopStreaming**(unsigned long chanNum);

Function: Stop streaming video on a channel if it is streaming

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Note 1: If the channel is also recording (writing encoded data to file) or decoding then this function does not stop MPEG recording or decoding.

If the channel is not recording nor decoding nor calling callback functions but only streaming encoded data then after calling this function the channel will not be encoding any more.

Note 2: If the channel is not streaming this function returns false.

MPEGIO2_API bool **MPEGIO2_noNetworkErrMsg**(unsigned long chanNum, bool stopErrMsg);

Function: Allow or disallow sending back network data receive error to application defined window (as passed at parameter **parentWnd** in function **MPEGIO2_initSDK**)

Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1

Input Parameter **stopErrMsg**:

true to stop sending more MPEGIO2_MSG_ERR_STREAM_XXX messages to application software, false (default) allow continuing sending this message when network read error happens during MPEG decoding using network stream as input.

Return true for success.

4.11 MPEG Decoding Functions

MPEG Decoding means playing back previously encoded MPEG data (can be encoded by **MPEGIO2** or other devices inc. software encoders) to display the decoded video on PC screen and/or on the "**MPEG Decode Output**" ports (RCA and S-Video sockets on the **Breakout Box** bottom layer). Note a channel cannot simultaneously encode and decode.

```
MPEGIO2_API bool  MPEGIO2_startMPEGDecoding(
                                    unsigned long chanNum,
                                    char *fileName,
                                    unsigned long videoStandard,
                                    unsigned short EncoderType,
                                    int MPEGStreamType,
                                    bool loopPlayFile = false,
                                    char *streamIPAddr = 0,
                                    int TCPPort = 1234,
                                    bool multicastStream = false);
```

Function: Start decoding an MPEG stream on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fileName**: The file name from which to decode mpeg stream, must be a valid file name including full path.
If this parameter is NULL then decoded data will come from a network address:
if parameter **multicastStream** is true, parameter **streamIPAddr** must contain a valid IP address, and parameter **TCPPort** must contain a valid port number, so that the decoded data will come from netweork address **streamIPAddr:TCPPort;**
if parameter **multicastStream** is false, **TCPPort** must contain a valid port number, so that the decoded data will come from netweork address **Local PC's IPAddress:TCPPort.**

Input parameter **videoStandard**: Video standard of the decoded MPEG data, must be 2 for PAL, or 1 for NTSC
Input parameter **EncoderType**: Must be 0 for Program Stream or 1 for Transport Stream
Input parameter **MPEGStreamType**: Must be one of:
0: MPEG4 Elementry Stream
1: MPEG4 Program Stream
2: MPEG4 Transport Stream
3: MPEG2 Program Stream
4: MPEG2 Transport Stream
5: MPEG1 System Stream
6: Video Elementry Stream
otherwise the type will be auto-detected (but could crash if data is an unknown type, or even MPEG1 type!)

Input parameter **loopPlayFile**: if true, and the **fileName** is not invalid, i.e., the decoding data is from a MPEG file, then will repeatedly play that file until specifically stopped by calling **MPEGIO2_stopMPEGDecoding**.
Input parameter **streamIPAddr**: If parameter **fileName** is NULL and parameter **multicastStream** is true, and this parameter contains a valid IP address in the form XXX.XXX.XXX.XXX then
a multicast socket will be created using this IP address and TCPPort parameter to receive data to be decoded.
Input parameter **TCPPort**: TCP Port number used to create a socket when parameter **fileName** is NULL.
Input parameter **multicastStream**: Only used when parameter **fileName** is NULL:
If true, a socket will be created in multicast mode using **streamIPAddr** and **TCPPort**,
If false, a unicast socket will be created using **TCPPort** and the local PC's IP address
Return true for success
Note: A channel cannot start decoding while is encoding or its MPEG decoder is currently powered down (call function **MPEGIO2_decoderPowerdownVal** to test).

```
MPEGIO2_API void  MPEGIO2_stopMPEGDecoding(unsigned long chanNum,
                                           unsigned __int64 *decodedSize = 0,
                                           unsigned __int64 *decodedStreamSize = 0);
```

Function: Stop decoding MPEG on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **decodedSize**: if not NULL, will receive the total decoded bytes

Output parameter **decodedStreamSize**: if not NULL, will receive the total decoded bytes that were streamed out

MPEGIO2_API bool **MPEGIO2_isMPEGDecoding**(unsigned long chanNum);
Function: Return true if a Channel is currently decoding
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number

MPEGIO2_API bool **MPEGIO2_pauseMPEGDecoding**(unsigned long chanNum, bool pause);
Function: Pause or resume MPEG decoding if a Channel is currently decoding
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **pause**: true to pause the decoding, false to resume the decoding if it is paused
Return true for success
Note 1: if the channel is not in decoding mode this function returns false
Note 2: If the channel is paused during decoding and parameter "**pause**" is true, or
if the channel is not paused during decoding and the parameter "**pause**" is false then
this function does not do anything and returns true

MPEGIO2_API bool **MPEGIO2_isMPEGDecodingPaused**(unsigned long chanNum);
Function: Return true if a channel is currently pausing its MPEG decoding (by calling
MPEGIO2_pauseMPEGDecoding(chanNum, true))
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.

MPEGIO2_API bool **MPEGIO2_setMPEGDecodingLoopPlayFile**(unsigned long chanNum,
bool loopPlayFile);
Function: Set if a Channel will loop play the decoded file repeatedly
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **loopPlayFile**: True means decoding on file will automatically repeat when reaching the file's end
Return true for success
Note 1: This function can succeed even if the channel is not currently decoding.
Note 2: By default decoding will not loop play.

MPEGIO2_API bool **MPEGIO2_getMPEGDecodingLoopPlayFile**(unsigned long chanNum);
Function: Return true if a Channel is set to loop play the decoded file repeatedly (the **loopPlayFile** parameter is true when
MPEGIO2_startMPEGDecoding/MPEGIO2_setMPEGDecodingLoopPlayFile was called)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

MPEGIO2_API bool **MPEGIO2_getDecodingFilePos**(unsigned long cardNum,
long &lower,
long &upper);
Function: Get the decoding MPEG file's current position in unit of bytes for an MPEGIO2 channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **lower**: the lower order part of the file's current byte position
Output parameter **upper**: the higher order part of the file's current byte position
Return true for success
Note: If the channel is not decoding this function returns false

MPEGIO2_API bool **MPEGIO2_decodingDataFromNetwork**(unsigned long chanNum,
bool &fromNetwork);
Function: Return if the MPEG decoding data is from incoming network streaming for an MPEG channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **fromNetwork**: True means decoded data is from incoming network stream, false(default) means decoding
data is from a file
Return true for success.

MPEGIO2_API bool **MPEGIO2_getDecodingFileName**(unsigned long chanNum,
char *fileName,
unsigned long fileNameLen);
Function: Return currently decoded file name for an MPEG channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **fileName**: memory chunk to receive the currently being decoded file name, must have at least

fileNameLen bytes space
Input parameter **fileNameLen**: the length of bytes of the memory pointed to by parameter **fileName**
Return true for success
Note 1: If **fileName** or **fileNameLen** is 0, or if the channel currently is not decoding this function returns false
Note 2: If the channel is decoding video from network stream the returned **fileName** contains NULL string (1st byte contains binary 0) but the function returns true
Note 3: If the decoded file name length is longer than **fileNameLen** then only **fileNameLen** - 1 characters are copied into **fileName** (the last character in **fileName** will be NULL)
Note 4: If the channel currently is not decoding the returned file name could be previously decoded file name, or meaningless if there was no previous decoding operation

```
MPEGIO2_API bool  MPEGIO2_getDecodeSizes(unsigned long chanNum,
                                         unsigned int &decodeTimeTotal,
                                         unsigned  __int64 &accumulatedDecodeSize,
                                         unsigned  __int64 &accumulatedDecodeStreamSize,
                                         char *fileName = 0,
                                         unsigned long fileNameLen = 0);
```

Function: Return the decoding sizes, time and file name etc for an MPEGIO2 Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Output parameter **decodeTimeTotal**: the time elapsed in decoding the MPEG file or incoming network video
Output parameter **accumulatedDecodeSize**: bytes that have been dencoded since decoding started
Output parameter **accumulatedDecodeStreamSize**: bytes that have been streamed out since decoding started
Output parameter **fileName**: if not NULL and **fileNameLen** is > 1, must point to a memory chunk at least **fileNameLen** bytes to receive the returned file name as the currently being decoded file name as a null-terminated string
Note if the decoding is being done on incoming network video then this parameter is ignored
Input parameter **fileNameLen**: if > 1, indicates the length in bytes of **fileName** parameter. If the current decoding file name length is longer than this length the returned file name string is truncated to be equal to this value as a null-terminated string.
Note this parameter is only useful if the decoded data is from file, not from incoming network stream.
Return true on success
Note: if the channel is not in decoding mode then the returned values in all output parameters might reflect the previous decoding operation's result, or they are meaningless if no previous decoding operation has ever happened.

```
MPEGIO2_API bool  MPEGIO2_decoderPowerdown(unsigned long chanNum, bool powerdown);
```

Function: Power-down/power-up the channel's MPEG decoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **powerdown**: True to power-down, false to power-up the decoder
Return true for success
Note 1: After power-down the decoder the channel cannot decode MPEG data any more until it's powered up again
Note 2: A channel in decoding mode cannot be powered down.

```
MPEGIO2_API bool  MPEGIO2_decoderPowerdownVal(unsigned long chanNum,
                                              bool &powerdown);
```

Function: Get power-down/power-up status of the channel's MPEG decoder
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Output parameter **powerdown**: True the decoder is powered-down, false is powered-up
Return true for success
Note: If the MPEG decoder is powered-down the channel cannot decode MPEG data until it's powered up again

```
MPEGIO2_API bool  MPEGIO2_decoderVideoDuringPause(unsigned long chanNum, int video);
```

Function: Set video displayed during MPEG decoder pause for the channel
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Input parameter **video**: 0 for displaying last frame(default) during pause, 1 for displaying blackness
Return true for success

```
MPEGIO2_API bool  MPEGIO2_decoderVideoDuringPauseVal(unsigned long chanNum,
                                                     int &video);
```

Function: Get video displayed during MPEG decoder pause for the channel

Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **video**: 0 for displaying last frame(default) during pause, 1 for displaying blackness
Return true for success

MPEGIO2_API bool **MPEGIO2_decoderAudioMute**(unsigned long chanNum, bool mute);
Function: Mute/un-mute the channel's MPEG decoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **mute**: True to mute, false to un-mute the decoder
Return true for success

MPEGIO2_API bool **MPEGIO2_decoderAudioMuteVal**(unsigned long chanNum, bool &mute);
Function: Return the mute/un-mute status of the channel's MPEG decoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **mute**: True the decoder is muted
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderAudioMode**(unsigned long chanNum, int mode);
Function: Set audio mode for the channel's MPEG decoder
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Input parameter **mode**:
0 -- Normal stereo, with both channel 1 and channel 2 on simultaneously.
1 -- Repeat channel 1 data on channel 2 (decoded channel 2 is discarded).
2 -- Repeat channel 2 data on channel 1 (decoded channel 1 is discarded).
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderAudioModeVal**(unsigned long chanNum, int &mode);
Function: Get audio mode of the channel's MPEG decoder
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Output parameter **mode**:
0 -- Normal stereo, with both channel 1 and channel 2 on simultaneously.
1 -- Repeat channel 1 data on channel 2 (decoded channel 2 is discarded).
2 -- Repeat channel 2 data on channel 1 (decoded channel 1 is discarded).
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderClockRecovery**(unsigned long chanNum, int cr);
Function: Set MPEG decoder Clock Recovery Mode for a Channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **cr**: must be:
For Transport Stream Encoder type:
0 -- Disable Clock recovery
1 -- Enable the low delay mode(Default).
2 -- Enable the buffer control mode (also called "circular buffer;").
3 -- Enable the low delay mode without checking for video/audio underflow condition due to data transfer jitter.
4 -- Enable the VCXO to control the audio PCM output frequency with the same clock recovery
For Program Stream Encoder type:
0 -- Disable Clock recovery
1 -- Enable the low delay mode(Default).
2 -- Enable the buffer control mode (also called "circular buffer).
3 -- Enable the low delay mode without checking for video/audio underflow condition due to data transfer jitter
Return True for success.
Note: This function can only be called when the channel is not decoding.

MPEGIO2_API bool **MPEGIO2_decoderClockRecoveryVal**(unsigned long chanNum, int &cr);
Function: Get MPEG decoder Clock Recovery Mode for a Channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **cr**: can be:
For Transport Stream Encoder type:
0 -- Disable Clock recovery
1 -- Enable the low delay mode(Default).
2 -- Enable the buffer control mode.
3 -- Enable the low delay mode without checking for video/audio underflow condition due to data transfer jitter.

4 -- Enable the VCXO to control the audio PCM output frequency with the same clock recovery
For Program Stream Encoder type:
0 -- Disable Clock recovery
1 -- Enable the low delay mode(Default).
2 -- Enable the buffer control mode.
3 -- Enable the low delay mode without checking for video/audio underflow condition due to data transfer jitter
Return True for success.

MPEGIO2_API bool **MPEGIO2_decoderBkColour**(unsigned long chanNum, int colour);
Function: Set background colour when the channel's MPEG decoder stops decoding MPEG video
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Input parameter **colour**: must be: 0x11 -- Normal operation with the last decoded picture(default)
0x31 -- Set display to black
0x51 -- Set display to blue
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderBkColourVal**(unsigned long chanNum, int &colour);
Function: Return the background colour when the MPEGIO2 channel's MPEG decoder stops decoding MPEG video
Input Parameter chanNum: MPEGIO2 Channel Number, must be between 0~totalChans-1
Output parameter colour: can be: 0x11 -- Normal operation with the last decoded picture(default)
0x31 -- Set display to black
0x51 -- Set display to blue
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderMpegCodingStandard**(unsigned long chanNum, int mpeg);
Function: Set decoder MPEG standard for the channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **mpeg**: must be: 1 or 1 or 4, for MPEG1, MPEG2, or MPEG4 respectively
Return true for success.
Note: This function can only be called when the channel is not decoding.

MPEGIO2_API bool **MPEGIO2_decoderMpegCodingStandardVal**(unsigned long chanNum, int &mpeg);
Function: Get the current decoder MPEG standard for the **MPEGIO2** channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **mpeg**: can be: 1 or 1 or 4, for MPEG1, MPEG2, or MPEG4 respectively
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderVideoFormat**(unsigned long chanNum, int vf);
Function: Set decoder Video standard (PAL/NTSC) for the channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **vf**: must be: 1 NTSC, 2 for PAL
Return true for success.
Note: This function can only be called when the channel is not decoding.

MPEGIO2_API bool **MPEGIO2_decoderVideoFormatVal**(unsigned long chanNum, int &vf);
Function: Get the current decoder Video standard (PAL/NTSC) for the channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **vf**: can be: 1 NTSC, 2 for PAL
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderMuxingType**(unsigned long chanNum, int st);
Function: Set decoder stream type for Muxed(Video+Audio), Video Only, or Audio Only Decoding for the channel
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Input parameter **st**: must be: 0: Video and Audio Stream Decoding
1: Video Elementry Stream Decoding
2: Audio Stream 1 Elementry Stream Decoding
3: Audio Stream 2 Elementry Stream Decoding
Return true for success.
Note: This function can only be called when the channel is not decoding.

MPEGIO2_API bool **MPEGIO2_decoderMuxingTypeVal**(unsigned long chanNum, int &st);
Function: Get MPEG decoder stream type: Muxed(Video+Audio), Video Only, or Audio Only Decoding for the channel
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Output parameter **st**: can be:
0: Video and Audio Stream Decoding
1: Video Elementry Stream Decoding
2: Audio Stream 1 Elementry Stream Decoding
3: Audio Stream 2 Elementry Stream Decoding
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderAutoDemux**(unsigned long chanNum, int ad);
Function: Set MPEG decoder auto demux mode for a channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **ad**: 0 = Manual Mode, 1 = Auto Mode (Default)

For Program Stream Demuxing:
0 -- Manual demux. User must specify which PES ID is video and which PES ID is audio.
1 -- Auto demux. The demux will automatically scan the input stream for PSM tables and automatically play the first program it finds.
For Transport Sream Demuxing:
0 -- Manual demux.
User must specify which PES ID is video and which PES ID is audio.
Manual demux mode also requires the other parameters to be set:
-- decoder Video PID
-- decoder Audio PID
-- decoder PCR PID
-- decoder Private PID if wishing to extract user data
1 -- Auto demux.
The demux will automatically scan the input stream for PSM tables and automatically play the first program it finds.
Return True for success.
Note: This function can only work when the channel is not currently decoding.

MPEGIO2_API bool **MPEGIO2_decoderAutoDemuxVal**(unsigned long chanNum, int &ad);
Function: Get MPEG decoder auto demux mode for a channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **ad**: 0 = Manual Mode, 1 = Auto Mode (Default)
Return True for success.

MPEGIO2_API bool **MPEGIO2_decoderMPEG4ShortHeader** (unsigned long chanNum, int header);
Function: Set decoding MPEG-4 short header and H.263 mode for a channel.
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **header**: 0 -- MPEG-4 only(Default), 1 -- MPEG-4 short header, 2 -- H.263
Return True for success.
Note: This function can only be called when the channel is not decoding.

MPEGIO2_API bool **MPEGIO2_decoderMPEG4ShortHeaderVal** (unsigned long chanNum, int &header);
Function: Get decoding MPEG-4 short header and H.263 mode for a channel.
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **header**: 0 -- MPEG-4 only(Default), 1 -- MPEG-4 short header, 2 -- H.263
Return True for success.

MPEGIO2_API bool **MPEGIO2_newVideoPID**(unsigned long chanNum, int newPID);
Function: Set Video PID for MPEG decoding for PS or TS Stream Encoder Type of a Channel:

For Program Stream Encoder Type:
This sets the Video PES ID to be demultiplexed.
In auto-demux mode, this command sets the initial Video PID. This initial Video PID will be valid until the arrival of the first PSM table. The demux will then replace the initial PID with new value taken from the PSM table.
In manual demux mode, this command sets the Video PID to be demuxed, which will remain valid indefinitely.
newPID range is 0 ~ 0xFF.

For Transport Stream Encoder Type:
This sets the video 1 stream PID.
newPID range is 0 ~ 0x1FFF.

Input Parameter **chanNum**: **MPEGIO2** Channel Number, must be between 0~totalChans-1
Input Parameter **newPID**: the new PID to set
Return True for success.
Note 1: This function can only work when the channel is not currently decoding
Note 2: This function does not automatically set Demux Auto Mode (use **MPEGIO2_decoderAutoDemux** to do that)

MPEGIO2_API bool **MPEGIO2_getVideoPID**(unsigned long chanNum, int &PID);
Function: Get Video PID for MPEG decoding for a Channel:
Input Parameter **chanNum**: **MPEGIO2** Channel Number, must be between 0~totalChans-1
Output Parameter **PID**: the PID received
Return True for success.

MPEGIO2_API bool **MPEGIO2_newAudio1PID**(unsigned long chanNum, int newPID);
Function: Set Audio 1 PID for MPEG decoding for PS or TS Stream Encoder Type of a Channel:
For PS Demuxing:
Parameter Range From 0 to 0xFF
This sets the Audio 1 PES ID to be demultiplexed.
In auto-demux mode, this command sets the initial Audio 1 PID. This initial Audio 1 PID will be valid until the arrival of the first PSM table. The demux will then replace the initial PID with a new value taken from the PSM table.
In manual demux mode, this command sets the Audio 1 PID to be demuxed, which will remain valid indefinitely.
Default Value is 0.

For TS Demuxing:
Parameter Range From 0 to 0x1FFF
This sets the audio 1 stream PID. Default Value is 1.
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input Parameter **newPID**: the new PID to set
Return True for success.
Note 1: This function can only work when the channel is not currently decoding
Note 2: This function does not automatically set Demux Auto Mode (use **MPEGIO2_decoderAutoDemux** to do that).

MPEGIO2_API bool **MPEGIO2_getAudio1PID**(unsigned long chanNum, int &PID);
Function: Get Audio 1 PID for MPEG decoding for a Channel
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output Parameter **PID**: the PID received
Return True for success.

MPEGIO2_API bool **MPEGIO2_newAudio2PID**(unsigned long chanNum, int newPID);
Function: Set Audio 2 PID for MPEG decoding for PS or TS Stream Encoder Type of a Channel:
For PS Demuxing:
Parameter Range From 0 to 0xFF
This sets the Audio 2 PES ID to be demultiplexed.
In auto-demux mode, this command sets the initial Audio 2 PID. This initial Audio 1 PID will be valid until the arrival of the first PSM table. The demux will then replace the initial PID with a new value taken from the PSM table.
In manual demux mode, this command sets the Audio 2 PID to be demuxed, which will remain valid indefinitely.
Default Value is 0.

For TS Demuxing:
Parameter Range From 0 to 0x1FFF
This sets the audio 2 stream PID. Default Value is 1.

Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input Parameter **newPID**: the new PID to set
Return True for success.
Note 1: This function can only work when the channel is not currently decoding

Note 2: This function does not automatically set Demux Auto Mode (use **MPEGIO2_decoderAutoDemux** to do that).

MPEGIO2_API bool **MPEGIO2_getAudio2PID**(unsigned long chanNum, int &PID);
Function: Get Audio 2 PID for MPEG decoding for a Channel:
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output Parameter **PID**: the PID received
Return True for success.

MPEGIO2_API bool **MPEGIO2_decoderSet_pcr_pid**(unsigned long chanNum, int pid);
Function: Set decoder PCR PID (program clock reference PID) for the channel if it is a Transport Stream Encoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **pid**: must be within 0 ~ 0x1FFF, default is 1
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderGet_pcr_pid**(unsigned long chanNum, int &pid);
Function: Get decoder PCR PID (program clock reference PID)for the channel if it is a Transport Stream Encoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **pid**: received PID
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderSet_private_data_pid**(unsigned long chanNum, int pid);
Function: Set decoder Private Data PID for the channel if it is a Transport Stream Encoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **pid**: must be within 0 ~ 0x1FFF, default is 8191 (0x1FFF)
Return true for success.

MPEGIO2_API bool **MPEGIO2_decoderGet_private_data_pid**(unsigned long chanNum, int &pid);
Function: Get decoder Private Data PID for the channel if it is a Transport Stream Encoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Output parameter **pid**: received PID
Return true for success

MPEGIO2_API bool **MPEGIO2_decoderSet_play_back_prog_num**(unsigned long chanNum,
int pnum);
Function: Set decoder Playback Program Number for the channel if it is a Transport Stream Encoder
Input Parameter **chanNum**: Channel Number, must be between 0~totalChans-1
Input parameter **pnum**: must be within 1 ~ (64 * 1024), default is 1
Return true for success.

4.12 Image Capturing Functions

```
MPEGIO2_API bool MPEGIO2_captureStill(  unsigned long chanNum,
                                        char *fileName,
                                        int fileFormat,
                                        unsigned long width,
                                        unsigned long height,
                                        char **imageBuf = 0,
                                        unsigned long cutoffBottomRows = 0,
                                        unsigned long DMARamMode = 0,
                                        int deinterlace = 0,
                                        bool pausePreview);
```

Function: Capture a frame of still image into file or buffer for an **MPEGIO2** channel on Windows XP/Windows 7 or above
Input Parameter **chanNum**: channel number, must be between 0 ~ totalChans - 1
Input Parameter **fileName**: full path name of the captured image file. If this is NULL then no still image file will be captured.
Input Parameter **fileFormat**: 0=BMP, 1=JPG, 2=GIF for WinXP, TIF for Win7 or above, 3=PNG
Input Parameter **width**: image width in pixel, must be <= 1024
Input Parameter **height**: image height in pixel, must be <= 576, for NTSC video set this to be <= 480
Input Parameter **imageBuf**: If not null then on successful function call, this receives the captured image in ARGB32 format
Input Parameter **cutoffBottomRows**: Pixel rows at the bottom of the captured image frame to be cut off, default is 0.

Input Parameter **DMARamMode**: For Windows 7 or above only, ignored for Windows XP:
1=Use Video Ram for DMA and lock the D3D surface when copying video frame data from driver to application
2=Use System Ram for DMA and unlock the D3D surface when copying video frame data from driver to application
Other value (default is 0)=SDK decides using 1 or 2 according to if the graphics card can do YUV to RGB conversion.
Note: on Intel graphics chipset (such as Q67 Express etc.) mode 2 must be used or the captured picture is garbled.

Input Parameter **deinterlace**: 0=None, 1=Odd Lines Copied to Even Lines, 2=Even Lines Copied to Odd Lines
Input Parameter **pausePreview**: For Windows 7 or above only: If true, will pause the video preview for all channels that are previewing before capturing the still then restore preview channels: useful only on some Intel Chipset(such as Q67) motherboards to prevent black/green stripes on captured image.
Return: true for success.
Note 1: Capture still can happen independent of MPEG encoding and video preview processes.
Note 2: The captured image's colour can be manually adjusted (without affecting colours in video preview and MPEG recording) by calling these functions described in the "**Video Preview Functions**" Section:
MPEGIO2_setBrightPorS, MPEGIO2_setContrastPorS, MPEGIO2_setSaturationPorS.
The current captured image's colours setup can also be retrieved from these functions described there**:**
MPEGIO2_getBrightPorS, MPEGIO2_getContrastPorS, MPEGIO2_getSaturationPorS.

4.13 Text & Graphics Overlay Functions

MPEGIO2 can overlay graphics and text on input video signal so that the encoded MPEG video will bear the overlaid graphics and text. The overlaid graphics/text will also appear on the video output ports through the **Loopback RCA** and **S-Video Output** sockets on the **Breakout Box.**

Overlays created by the SDK users are organized as "Overlay Items": each has item number, colour, X/Y position, Alpha, Blink, and many other properties. "Overlay Items" can be created, deleted, moved, shown, hidden, redrawn by calling relevant SDK functions. Between **MPEGIO2** SDK exit and re-start, application software created Overlay Items will be saved to and read out from the **initFile**, therefore previously created Overlay Items can be reused.

Each **MPEGIO2** channel has 4 special "Box Overlay" items that have their own colour, alpha transparency properties un-related to other text & graphics overlays: these 4 Box Overlay Items can be used as partially-transparent backgrounds to highlight text and graphics overlays.

Non-Box Overlay Items have Text String, Graphics File, Timer /Counter, and Rectangle item types, that can simultaneously use 252 different colours, through a Colour Lookup Table (CLUT): entries in this CLUT can be either loaded completely from a graphics file by calling function "**MPEGIO2_loadCLUTFromFile**", or loaded manually a-few-entries a time by calling function "**MPEGIO2_loadCLUT**". Several levels of Alpha transparency and Blink property can also be applied to text & graphics overlay items.

The **MPEGIO2** SDK defines Overlay Items Types as:

OT_TEXT	=	0,
OT_TIMER	=	1,
OT_RECTANGLE	=	2,
OT_BOX	=	3,
OT_GRAPHICS	=	4.

4.13.1 Text & Graphics Overlay Items

```
MPEGIO2_API bool  MPEGIO2_setOverlayGraphics(
                        unsigned long chanNum,
                        HWND wnd, char *fileName,
                        HBITMAP bitmap,
                        unsigned long x,
                        unsigned long y,
                        unsigned long &width,
                        unsigned long &height,
                        bool usePalette = true,
                        bool transparent = true,
                        unsigned long transparencyColour = 0,
                        bool alpha = false,
                        bool blink = false,
                        bool *graphicsOPComplete = 0);
```

Function: Load a graphics file "**fileName**" into **MPEGIO2** channel as overlay
Input parameter **chanNum**: the channel num, must be between 0 ~ (totalChans - 1)
Input parameter **wnd**: window to receive SDK messages sent back to application program calling this function: if NULL or not a valid window handle then no msg will be sent back from SDK to caller application.
Input parameter **fileName**: if not NULL, must points to a valid grapgics file of 8-bit colour, DIB Section. If this parameter and parameter bitmap are both NULL, a file selection dialog will appear asking for a graphics file to be selected to be used as the bitnap source
Input parameter **bitmap**: if **fileName** is NULL or invalid and this parameter points to a valid Windows SDK bitmap handle, then this bitmap will be used to load onto video frame.
Note if this parameter is non-null and represents a valid Windows SDK HBITMAP, its memory must NOT be released by the calling application until **MPEGIO2 SDK** is ended, or unless this overlay is deleted. This is because as long as this graphics overlay is available, **MPEGIO2** SDK will be using the bitmap object pointed to by this parameter to display overlay: **MPEGIO2** SDK does not create a local copy of the bitmap resource represented by this parameter, this also implies if the calling application changed the content of this bitmap after initially calling this function successfully, redrawing the overlay created by the initial call to this function will see the new bitmap content being displayed on **MPEGIO2** video frame.
Note, in VB or C#, use their Bitmap Class' GetHbitmap method to get the HBITMAP handle of a bitmap then pass that to this function.
Input parameter **x/y/width/height**: Specify the overlaid graphics position and width/height on video frame, in pixel unit. On input**, width/height** can be zeroes which means the graphics bitmap's actual width/height will be used to overlay the bitmap on video frame. On successful return (TRUE) the width and height will hold the graphics bitmap's actual width and height.
Input parameter **usePalette**: true to use the loaded graphics file's palette as OSD IC's overlay colour palette
Input parameter **transparent**: if true, then any pixel on the graphics bitmap having colour "**transparencyColour**" will expose the video pixel beneath it
Input parameter **transparencyColour**: only useful when parameter "**transparent**" is true, default is black (RGB=(0,0,0))
Input parameter **alpha**: if true, the graphics will have alpha value as set by function **MPEGIO2_setOverlayAlpha**()
Input parameter **blink**: if true, the graphics will have blinking value as set by function **MPEGIO2_setOverlayBlink**()
Output parameter **graphicsOPComplete**: if not NULL, points to a bool variable address that will be set to true only when the overlay graphics operation is completed

Return true for success.
Note: Loading graphics to Overlay IC on board the **MPEGIO2** card is an asynchronous process:
when this function returns true, it does not necessarily mean the graphics file loading process is finished: only when a message MPEGIO2_MSG_OK_SET_OSDGRAPHICS / MPEGIO2_MSG_ERR_SET_OSDGRAPHICS is received by the called application then the graphics file loading is really finished (succeeded or failed).

```
MPEGIO2_API bool  MPEGIO2_setOverlayText(unsigned long chanNum,
                                          HWND wnd,
                                          char *textOrFile,
                                          wchar_t *textW = 0,
                                          bool isTextFile = false,
                                          unsigned short x = 0,
                                          unsigned short y = 0,
                                          bool useDLF = false,
                                          unsigned char bkMode = TRANSPARENT,
                                          COLORREF fcolour = DEFAULT_FCOLOUR,
                                          COLORREF bcolour = DEFAULT_BCOLOUR,
                                          int width = 24,
                                          int height = 24,
                                          int weight = 400,
                                          unsigned char italic = 0,
                                          char *typeFace = DEFAULT_TYPE_FACE,
                                          unsigned char alpha = 0,
                                          unsigned char blink = 0);
```

Function: Display text string overlay on a channel's video input
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **wnd**: window handle that will receive messgaes sent from SDK to application program, if NULL no msg will be sent back from SDK.
Input parameter **textOrFile**: If **isTextFile** == false, this must point to a NULL-terminated ASCII text string with printable characters (ASCII code 0x20 ~ 0x7F).
If **isTextFile** == true, this must point to a pointer to a valid text file name inc. path.
The string pointed to by this parameter must be <= 512 bytes inc. the terminating NULL.
Input parameter **textW**: When **textOrFile** is NULL and **textW** is not NULL, the wide character null-terminated string (Unicode String) pointed to by **textW** will be used as the overlay text to display on video: in this case the font represented by **typeFace** parameter should also be a Unicode Font.
When **textOrFile** is not null **textW** will be ignored and the ASCII string pointed to by **textOrFile** will be used.
Input parameter **isTextFile**: If the parameter **textOrFile** points to a text string (when this parameter is false) or a text file name (when this parameter is true).
Input parameter **x**: horizontal position of the text string to appear on the video frame, 0~718
Input parameter **y**: vertical position of the text string to appear on the video frame, 0~574 for PAL, 0~478 for NTSC
Input parameter **useDLF**: Only meaningful when isTextFile == false(since text file always uses downloaded font)
If to use downloadable font to display the text string:
if true and there is a downloaded font available which matches the text's font description then that downloaded font will be used. If there is no downloaded font matching the text's font description then a font will be downloaded.
If false, no font will be downloaded, text will be displayed using newly created font character bitmaps.
Input parameter **bkMode**: the text display's background mode: 1= TRANSPARENT(default), 2= OPAQUE
Input parameter **fcolour**: foreground colour in RBG format for the text, default is yellow
Input parameter **bcolour**: background colour in RBG format for the text background, only meaningful when **bkMode**== OPAQUE, default is white
Input parameter **width**: font width, default is 24, same meaning as in LOGFONT.lfWidth of the Windows SDK
Input parameter **height**: font height, default is 24, same meaning as in LOGFONT.lfHeight of the Windows SDK
Input parameter **weight**: The weight of the font in the range 0 through 1000, same meaning as in LOGFONT.lfWeight of the Windows SDK, default is 400
Input parameter **italic**: font italic, default is 0 (no italic), non-zero means italic
Input parameter **typeFace**: type face of the text string: The length of this string must not exceed 32 TCHAR values, including the terminating NULL.
Input parameter **alpha**: set new alpha value for all overlay display items:

default is 0: do not change alpha value for overlay display.
Other valid values for **alpha**: 1=50% transparency, 2=75% transparency, 3=25% transparency.
Higher **alpha** means the overlay has more visibility against its background video.

Input parameter **blink**: set new blink interval value for all overlay display items:
default is 0: do not change blink value for overlay display
Other valid values for **blink**: 1 = 0.25 second, 2 = 0.5 second, 3 = 1 second, 4 = 2 second.
Higher blinking interval value means the blinking is slower

Return: true for success

Note 1: Setting text string to Overlay IC on board the **MPEGIO2** card is an asynchronous process:
when this function returns with true, it does not necessarily mean the text string setting process is finished:
only when a message MPEGIO2_MSG_OK_SET_OSDTEXT (or MPEGIO2_MSG_ERR_SET_OSDTEXT) is received by the calling application then the text string setting is really finished or failed.

Note 2: As defined in **MPEGIO2.h**:

```
#define DEFAULT_FCOLOUR     RGB(255, 0, 0) // Red
#define DEFAULT_BCOLOUR     RGB(255, 255, 255) // White
#define DEFAULT_TYPE_FACE  "Times New Roman" // Default Font Type Face.
```

Note 3: If the width of the text string is beyond the right edge of the video frame then the text string will be chopped off the extra characters to fit inside the video frame; if the 1st character is off the right edge of the video frame then the text overlay will fail and message MPEGIO2_MSG_ERR_SET_OSDTEXT will be sent to parentWnd which was passed to **MPEGIO2_initSDK** as parameter.

Note 4: If the height of the text string is beyond the bottom edge of the video frame then the text string will fail to be overlaid and message MPEGIO2_MSG_ERR_SET_OSDTEXT will be sent to parentWnd that was passed to **MPEGIO2_initSDK** as parameter.

```
MPEGIO2_API bool  MPEGIO2_getOverlayText(
                              unsigned long chanNum,
                              unsigned short itemNum,
                              unsigned short textSize,
                              char **text = 0,
                              wchar_t **textW = 0,
                              unsigned short *x = 0,
                              unsigned short *y = 0,
                              bool *useDLF = 0,
                              unsigned char *bkMode = 0,
                              unsigned long *fcolour = 0,
                              unsigned long *bcolour = 0,
                              int *width = 0,
                              int *height = 0,
                              int *weight = 0,
                              unsigned char *italic = 0,
                              char **typeFace = 0,
                              unsigned char **alpha = 0,
                              unsigned char *blink = 0);
```

Function: Retrieve a text overlay item on a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Input parameter **itemNum**: the index number of the overlay item which has type==OT_TEXT (0).
Input parameter **textSize**: the length of memory pointed to by parameter "***text**", must be > 1 if "**text**" is not NULL.
Output parameter **text**: If not NULL, ***text** point to memory >= **textSize** long in bytes to receive the item's ASCII text string.
Output parameter **textW**: If not NULL, ***textW** point to memory >= **textSize** long in Unicode characters to receive the item's Unicode text string.
Output parameter **x**: get the horizontal position of the text string to appear on the video frame, 0~718
Output parameter **y**: get the vertical position of the text string to appear on the video frame, 0~574 for PAL, 0~478 for NTSC
Output parameter **useDLF**: if to use downloadable font to display the text string:
if returned true and there is a downloaded font available which matches the text's font description then that downloaded font will be used. If there is no downloaded font matching the text's font description then a font will be downloaded.
Using downloaded font is faster in displaying the text although downloading the font takes time.
If false, no font will be downloaded, text will be displayed using newly created font character bitmaps.
Output parameter **bkMode**: return if the text display's background mode is transparent(1) or opaque(2)

Output parameter **fcolour**: return the foreground colour in RBG format for the text, default is red
Output parameter **bcolour**: return the background colour in RBG format for the text background, only meaningful when bkMode== OPAQUE(2), default is white
Output parameter **width**: return the font width, same meaning as in LOGFONT.lfWidth of the Windows SDK
Output parameter **height**: return the font height, same meaning as in LOGFONT.lfHeight of the Windows SDK
Output parameter **weight**: return the The weight of the font in the range 0 through 1000, same meaning as in LOGFONT.lfWeight of the Windows SDK, default is 400
Output parameter **italic**: return the font italic, 0 means no italic, non-zero means italic
Output parameter **typeFace**: If not NULL, ***typeFace** will get the type face of the text string: The length of this memory chunk must be >= 32 bytes, including the terminating NULL.
Output parameter **alpha**: return the alpha value for all overlay display items: default is 0: no alpha,
valid values for alpha: 1=50% transparency, 2=75% transparency, 3=25% transparency.
Higher alpha means the overlay has more visibility against its background video.
Output parameter **blink**: return the blink interval value for all overlay display items: default is 0: no blink
valid values for blink: 1 = 0.25 second, 2 = 0.5 second, 3 = 1 second, 4 = 2 second.
Higher blinking interval value means the blinking is slower.
Return: true for success.
Note 1: If any of the output parameter is supplied as NULL then that parameter is ignored
Note 2: If the item with index num "itemNum" does not exist or is not type OT_TEXT (0) then this function returns false.

MPEGIO2_API unsigned short **MPEGIO2_get1stTextOverlayNum** (unsigned long chanNum);
Function: Return the first text overlay item's number.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return: If this channel has some overlay item as text (type is OT_TEXT) return the 1st text overlay item's index in the overlay item list, otherwise return -1, any other failure also return -1 (0xFFFF).

MPEGIO2_API bool **MPEGIO2_setOverlayTimer**(
unsigned long chanNum,
HWND wnd,
unsigned short x = 0,
unsigned short y = 0,
unsigned char timerFormat = 0,
int startValue = 1,
int stopValue = 100,
int stepValue = 1,
bool eraseOnStop = false,
bool HMSDisplay = false,
unsigned char bkMode = TRANSPARENT,
COLORREF fcolour = DEFAULT_FCOLOUR,
COLORREF bcolour = DEFAULT_BCOLOUR,
int width = 24,
int height = 24,
int weight = 400,
unsigned char italic = 0,
char *typeFace = DEFAULT_TYPE_FACE,
unsigned char alpha = 0,
unsigned char blink = 0,
unsigned long timerFreq = 3);
Function: Display timer overlay on a channel's video input
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **wnd**: window handle that will receive messgaes sent from SDK to application program
Input parameter **x**: horizontal position for the timer string to appear on the video frame, 0~718
Input parameter **y**: vertical position for the timer string to appear on the video frame, 0~574 for PAL, 0~478 for NTSC
Input parameter **timerFormat**: 0=Time Only(default), 1=Time and Date, 2=Counter
Input parameter **startValue/stopValue/stepValue**: Only meaningful when parameter timerFormat=2: the start,stop and step values for counter. The unit for each step==1 is 100ms. However, if HMSDisplay==true then the step's unit will be one second
Input parameter **eraseOnStop**: Only meaningful when parameter timerFormat=2: if true, erase the counter display when

timer stops

Input parameter **HMSDisplay**: Only meaningful when parameter timerFormat=2: if true, display the counter in Hour:Minute:Second format, if false, display the counter as number

Input parameter **bkMode**: the timer display's background mode: 1 = TRANSPARENT(default), 2 = OPAQUE

Input parameter **fcolour**: foreground colour in RBG format for the timer, default is red

Input parameter **bcolour**: background colour in RBG format for the timer background, only meaningful when bkMode== OPAQUE, default is white

Input parameter **width**: font width, default is 24, same meaning as in LOGFONT.lfWidth of the Windows SDK

Input parameter **height**: font height, default is 24, same meaning as in LOGFONT.lfHeight of the Windows SDK

Input parameter **weight**: The weight of the font in the range 0 through 1000, same meaning as in LOGFONT.lfWeight of the Windows SDK, default is 400

Input parameter **italic**: font italic, default is 0 (no italic), non-zero means italic

Input parameter **typeFace**: type face of the timer string: The length of this string must not exceed 32 TCHAR values, including the terminating NULL.

Input parameter **alpha**: set new alpha value for all overlay display items:
default is 0: do not change alpha value for overlay display
Other valid values for alpha: 1=50% transparency, 2=75% transparency, 3=25% transparency.
Higher alpha means the overlay has more visibility against its background video.

Input parameter **blink**: set new blink interval value for all overlay display items:
default is 0: do not change blink value for overlay display
Other valid values for blink: 1 = 0.25 second, 2 = 0.5 second, 3 = 1 second, 4 = 2 second.
Higher blinking interval value means the blinking is slower.

Input parameter **timerFreq**: Valid values 1 ~ 10 to indicate the Overlay Timer Update time in milli seconds:
1 means every 100ms the timer overlay is redrawn,
2 means every 200ms, ... 10 means every 1000ms (1sec.) the timer overlay is redrawn,
default is 3: every 300ms the timer overlay gets updated.

Return: true for success. Since each channel can only have at most one timer at any time, if the channel already has a timer created previously then this function will change the current timer's parameters according to this function supplied new parameters and returns true.

Note 1: Setting timer to Overlay device on board the MPEGIO2 channel is an asynchronous process: when this function returns true, it does not necessarily mean the timer setting process is finished: only when a message MPEGIO2_MSG_OK_SET_OSDTIMER (or MPEGIO2_MSG_ERR_SET_OSDTIMER) is received by the caller application then the timer setting really succeeded or failed.

Note 2: timer overlay always uses its own font constructed from the parameters of this function call as downloaded font.

Note 3: If the timer string's height is beyond the video frame's bottom edge or the timer string's first character is beyond the video frame's right edge then the timer will fail to load and message MPEGIO2_MSG_ERR_SET_OSDTIMER will be sent to parentWnd that was passed to **MPEGIO2_initSDK** as parameter.

MPEGIO2_API bool **MPEGIO2_getOverlayTimer**(unsigned long chanNum,
unsigned short &overlayNum,
unsigned char &timerFormat);

Function: Get the timer overlay status on a channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Output Parameter **overlayNum**: if the timer overlay has been defined, this will hold its overlay index number otherwise this will not change

Output parameter **timerFormat**: if the timer overlay has been defined, this will hold its format:
0=Time Only, 1=Time and Date, 2=Counter

Return true if the timer overlay exists, false for not exists or failure

MPEGIO2_API bool **MPEGIO2_startStopTimerCounter**(unsigned long chanNum, bool start);

Function: Start or stop the already created counter type timer on a channel

Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number

Input parameter **start**: true to start counter, false to stop counter

On success, return true.

Note: if the counter has been started and parameter "**start**" is true,
or if the counter has been stopped (or never started) and parameter "**start**" is false, this function returns false.

MPEGIO2_API bool **MPEGIO2_isCounterStarted**(unsigned long chanNum);

Function: Return if the counter type timer has been started (during counting) on a channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

On success, return true if the channel has a timer of counter type and is currently displaying the counting.

```
MPEGIO2_API bool MPEGIO2_hasTimer(unsigned long chanNum,
                                  unsigned char *timerFormat,
                                  unsigned short *timerItemNum);
```

Function: Return if the channel has a timer overlay item defined
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **timerFormat**: if not NULL and the channel has a timer overlay defined this will hold the timer type:
0=time only, 1=date+time, 2=counter
Output parameter **timerItemNum**: if not null and the function returns true, this will get the timer overlay item's number
On success, return true if the channel has a timer.

```
MPEGIO2_API bool MPEGIO2_hasCounter(unsigned long chanNum,
                                    unsigned short *counterItemNum);
```

Function: Return if the channel has a timer overlay item defined and its timerFormat is Counter (2)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **counterItemNum**: if not null and the function returns true, this will get the timer overlay item's number
On success, return true if the channel has a timer and the timer's format is counter (2).

```
MPEGIO2_API LOGFONT *MPEGIO2_getCurrLFont(unsigned long chanNum);
```

Function: Retrieve the pointer of the LOGFONT structure used for a channel to create Text/Timer/Tex File Overlay Items
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
On success, return the pointer to the LOGFONT structure, returns 0 for failure.
Note: "LOGFONT " is defined in Windows SDK file WinGDI.h.

```
MPEGIO2_API unsigned long MPEGIO2_setOverlayRectangle(unsigned long chanNum,
                                                      unsigned short x,
                                                      unsigned short y,
                                                      COLORREF colour,
                                                      int width,
                                                      int height,
                                                      unsigned char alpha = 0,
                                                      unsigned char blink = 0);
```

Function: Add a rectangle overlay to a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **x**: the rectangle's horizontal position on video frame in pixle unit, starting from 0, maximum is 718
Input parameter **y**: the rectangle's vertical position on video frame in pixle unit, starting from 0,
maximum is 574 for PAL, 478 for PAL
Input parameter **colour**: the colour used to display the rectangle
Input parameter **width**: width of the rectangle in pixel unit, minimum 2
Input parameter **height**: height of the rectangle in pixel unit, minimum 2
Input parameter **alpha**: set new alpha value for all overlay display items:
default is 0: do not change alpha value for overlay display
other valid values for alpha: 1=50% transparency, 2=75% transparency, 3=25% transparency.
Higher alpha means the overlay has more visibility against its background video.
Input parameter **blink**: set new blink interval value for all overlay display items:
default is 0: do not change blink value for overlay display
Other valid values for blink: 1 = 0.25 second, 2 = 0.5 second, 3 = 1 second, 4 = 2 second.
Higher blinking interval value means the blinking is slower
If successful return unsigned long with its low 16 bits containg the newly added overlay item's number in overlay item list for this channel, its high 16-bit containing the total number of added overlays for this channel
Note: **x+width** must < 720, **y+height** must < 576 for PAL, must < 480 for NTSC, otherwise the function will fail

```
MPEGIO2_API bool MPEGIO2_loadCLUT( unsigned long chanNum,
                                   unsigned char startIndex,
                                   unsigned char len,
                                   unsigned char *RGB,
                                   bool alpha, bool blink);
```

Function: Load Colour Look Up Table (CLUT) entries for Non-Box overlays on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **startIndex**: index of the first entry in CLUT the load the data to, must be between [0, 255] inclusive.
Input Parameter **len**: number of entries in the CLUT to load data, must be between [1, 255] inclusive.
Input Parameter **RGB**: An array of 3 elements per row, each row represents the RGB colour bytes for one pixel.
Must be >= len * 3 bytes long.
Input Parameter **alpha**: True for enabling Alpha for the loaded entries.
Input Parameter **blink**: True for enabling Blink for the loaded entries.
Return true for success
Note: the CLUT is max. 256 entries long, but the last entry is used for transparant colour so is not loadedable here.

MPEGIO2_API bool **MPEGIO2_loadCLUTFromFile**(unsigned long chanNum, char *fileName);
Function: Load Colour Look Up Table (CLUT) entries from a graphics file for Non-Box overlays on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fileName**: must be a valid graphics file name inc. path,
the file must be 8-bit colour (256 colours), indexed, and DIBSectioned.
Return true for success
Note 1: the CLUT has max. 256 entries, but the last entry is used for transparant colour so is not loadedable by this function
Note 2: All entries loaded this way will have no alpha nor blink.

MPEGIO2_API bool **MPEGIO2_displayColourPalette**(unsigned long chanNum,
bool usePALFormat = true,
HWND wnd = false);
Function: Display current Colour Look Up Table (CLUT) (256 colours) on video frame for an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **usePALFormat**: true to use PAL video frame height (576 pixel),
false to use NTSC video frame height (480 pixel)
Input parameter **wnd**: parent window on which a dialog window appears after displaying the CLUT to allow user to end the display and return back to normal operation. If NULL, the desktop window will be used as parent window
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOverlayAlpha**(unsigned long chanNum, unsigned char alpha);
Function: Set alpha value for all overlays (except Box overlays) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input Parameter **alpha**: the new alpha value, between [1, 3] inclusive.
= 1 for 50% alpha, = 2 for 75% alpha, = 3 for 25% alpha.
Larger value has more visibility for the overlay items.
Return true for success
Note : This function simply changed the overall overlay alpha value for a channel when it creats new overlay items:
this function does not update all existing overlay items' alpha display until each item is redrawn.

MPEGIO2_API bool **MPEGIO2_getOverlayAlpha**(unsigned long chanNum, unsigned char &alpha);
Function: Get alpha value for all overlay items (except Box overlays) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **alpha**: Current alpha value, [1, 3] inclusive. 1 for 50% alpha, 2 for 75% alpha, 3 for 25% alpha.
Return true for success.

MPEGIO2_API bool **MPEGIO2_setOverlayBlink**(unsigned long chanNum, unsigned char blink);
Function: Set blinking intervals for all overlays (except Box overlays) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **blink**: blinking intervals: 0 = 0.25 second, 1 = 0.5 second, 2 = 1 second, 3 = 2 second.
Return true for success
Note: This function simply changed the overall overlay blink value for a channel when it creats new overlay items:
this function does not update all existing overlay items' blink display until each item is redrawn.

MPEGIO2_API bool **MPEGIO2_getOverlayBlink**(unsigned long chanNum, unsigned char &blink);
Function: Get blinking intervals for all overlay items (except Box overlays) on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output Parameter **blink**: blinking intervals: 0 = 0.25 second, 1 = 0.5 second, 2 = 1 second, 3 = 2 second.
Return true for success

MPEGIO2_API unsigned short **MPEGIO2_setOverlayItem**(unsigned long chanNum, overlayItemParam *ci);

Function: Modify an overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input/Output parameter **ci**: pointer to a current overlay item structure(**overlayItemParam** is defined in **MPEGIO2.h** file) describing an existing item's parameters:To modify an existing overlay item, the caller must set the itemNum component of this parameter to an existing overlay item's number. If this parameter is NULL or if there is no existing item with its number = to **ci**->ItemNum, this function returns 0 without doing anything, otherwise that overlay item's various parameters will be modified according to this parameter's fields.
Return: On success, non-zero as the total overlay item numbers for this channel, return zero for failure or this channel has no overlay item with index number == ci->itemNum.

MPEGIO2_API unsigned short **MPEGIO2_getOverlayItem**(unsigned long chanNum,
overlayItemParam *item);

Function: Retrieve an overlay item's parameters on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input/Output parameter **item**: pointer to the overlay item structure describing the item parameters, the caller must set the **itemNum** component of this parameter to indicate an existing overlay item number: If this parameter is NULL this function returns 0 without doing anything, otherwise if there is an overlay item with index number == item->itemNum then that overlay item's various parameters will be assigned to this parameter's fields and this function returns the total overlay items for the channel.
Return non-zero as the total overlay item numbers for this channel, zero for failure or this channel has no overlay item.
Note 1: Structure "**overlayItemParam**" is defined in file **MPEGIO2.h**.
Note 2: The pointer "**item**" passed to .Net languages doesn't seem to get assigned with values so don't use this function in .Net environment programming.

MPEGIO2_API bool **MPEGIO2_setOverlayItemFColour**(unsigned long chanNum,
unsigned short itemNum,
unsigned long fcolour,
bool redraw = true);

Function: Set the foreground colour of an overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **itemNum**: the overlay item's index number in overlay list for this channel
Output parameter **fcolour**: the new foreground colour of the overlay item in ARGB format(Alpha byte is not used).
Returns true for success.

MPEGIO2_API bool **MPEGIO2_setOverlayItemBColour**(unsigned long chanNum,
unsigned short itemNum,
unsigned long bcolour,
unsigned char bkMode,
bool redraw = true);

Function: Set the background colour and background mode of an overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **itemNum**: the overlay item's index number in overlay list for this channel
Input parameter **bcolour**: the new background colour of the overlay item in ARGB format(Alpha byte is not used).
Input parameter **bkMode**: new background mode: 1=Transparent, 2=Opaque
Input parameter **redraw**: if true, redraw the overlay item
Returns true for success.

MPEGIO2_API bool **MPEGIO2_setOverlayItemFont**(unsigned long chanNum,
unsigned short itemNum,
unsigned long fcolour,
int width,
int height,
int weight,
unsigned char italic,
char *typeFace,
bool redraw = true);

Function: Change one overlay item's font parameters on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **itemNum**: the index number of the overlay item which has
type==OT_TEXT (0) or OT_TIMER(1) or OT_TEXT_FILE(5)
Input parameter **fcolour**: the new foreground colour of the overlay item
Input parameter **width**: the font width, default is 24, same meaning as in LOGFONT.lfWidth of the Windows SDK
Input parameter **height**: the font height, default is 24, same meaning as in LOGFONT.lfHeight of the Windows SDK
Input parameter **weight**: the weight of the font in the range 0 through 1000, same meaning as in LOGFONT.lfWeight of the Windows SDK, default is 400
Input parameter **italic**: the font italic, default is 0 (no italic), non-zero means italic
Input parameter **typeFace**: If not NULL, pass the type face name of the text string: The length of this memory chunk must be <= 32 bytes, including the terminating NULL.
Input parameter **redraw**: if true, the item will be redrawn with the newly changed font parameters
Return: true for success
Note : If the item does not exist, or is not type OT_TEXT, OT_TIMER or OT_TEXT_FILE, this function returns false.

```
MPEGIO2_API bool  MPEGIO2_setOverlayItemAlphaBlink(unsigned long chanNum,
                                                    unsigned short itemNum,
                                                    unsigned char alpha,
                                                    unsigned char blink,
                                                    bool redraw = true);
```

Function: Change an overlay item's alpha and blink on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **itemNum**: the overlay item's index number in overlay list for this channel
Input parameter **alpha**: the new alpha value: 0 to indicate this item has no alpha, or
the alpha value(= 1 for 50% alpha, = 2 for 75% alpha, = 3 for 25% alpha)
Input parameter **blink**: the new blink value: 0 to indicate this item has no blink, or
1 = 0.25 second, 2 = 0.5 second, 3 = 1 second, 4 = 2 second
Input parameter **redraw**: if true, redraw the overlay item
Returns true for success.

4.13.2 The Box Overlay Items

```
MPEGIO2_API unsigned long MPEGIO2_setOverlayBox(
                                    unsigned long chanNum,
                                    unsigned short x,
                                    unsigned short y,
                                    unsigned short width,
                                    unsigned short height,
                                    unsigned char &boxNum,
                                    bool hasBoundary = true,
                                    unsigned char boxColour = 0,
                                    unsigned char alpha = 0);
```

Function: Display a Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **x**: Box's horizontal position on video frame in pixel unit, must within 0 ~ 719
Input parameter **y**: Box's vertical position on video frame in pixel unit,
must within 0 ~ 575 for PAL, within 0 ~ 479 for NTSC
Input parameter **width**: Box' width in pixel unit, must be within 1~720
Input parameter **height**: Box' height in pixel unit, must be within 1~576 for PAL, 1~480 for NTSC
Output parameter **boxNum**: On success, the newly created Box overlay's number, always within [0, 3] inclusive.
Input parameter **hasBoundary**: if true the box has a white boundary
Input parameter **boxColour**: Box's colour:must be within [0, 15] inclusive:
0 White (75% Amplitude 100% Saturation) (default),
1 Yellow (75% Amplitude 100% Saturation),
2 Cyan (75 % Amplitude 100 Saturation),
3 Green (75% Amplitude 100% Saturation),
4 Magenta (75% Amplitude 100% Saturation),
5 Red (75% Amplitude 100% Saturation),

6 Blue (75% Amplitude 100% Saturation),
7 0% Black,
8 100% White,
9 50% Gray,
10 25% Gray,
11 Blue (75% Amplitude 75% Saturation),
12 Box CLUT entry 0 colour, see function **MPEGIO2_setBoxCLUT**,
13 Box CLUT entry 1 colour, see function **MPEGIO2_setBoxCLUT**,
14 Box CLUT entry 2 colour, see function **MPEGIO2_setBoxCLUT**,
15 Box CLUT entry 3 colour, see function **MPEGIO2_setBoxCLUT**.

Input parameter **alpha**: 0= this Box item has No Alpha(solid),
other values set the alpha for all Box Overlay items:
1 for 50% alpha, 2 for 75% alpha, 3 for 25% alpha.
Note Box overlay's alpha is not affected by non-box overlay items' alpha settings.

If successful return unsigned long with its low 16 bits containg the newly added overlay item's number in overlay item list for this channel, its high 16-bit containing the total number of added overlays for this channel

Return 0 for failure.

Note 1: x+width must be < 720, y+height must be < 576 for PAL, < 480 for NTSC

Note 2: Box Overlay items have display priority over all other Non-Box overlay items:
when Box overlay items and non-box overlay items overlap, the Box overlay items appear in front of the non-box overlay items in the overlapped area.

MPEGIO2_API bool **MPEGIO2_hasUnusedOverlayBox**(unsigned long chanNum,
unsigned char *boxNum);

Function: return if a channel still has unsed singleBox overlay
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **boxNum**: if there is some unused overlay box and this parameter is not NULL then one un-used overlay box number(within 0~3) is returned in this parameter
Return true for still having un-used overlay box, false for all 4 overlay boxes are already defined.
Note: when first initialized, each channel has 4 overlay boxes with their box numbers from 0 to 3

MPEGIO2_API bool **MPEGIO2_getOverlayBox**(unsigned long chanNum, unsigned char boxNum);

Function: Get a Box overlay status on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the Box number, between [0, 3] inclusive.
Returning true means the Box overlay exists, false means not exists or failure.

MPEGIO2_API bool **MPEGIO2_setBoxEnable**(unsigned long chanNum,
unsigned char boxNum,
bool enable);

Function: Enable/Disable an existing Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Input Parameter **enable**: true for enable, false for disable
Return true for success
Note: When enabled (default) the Box Overlay appears, when disabled it disappears from the video surface but is not deleted.

MPEGIO2_API bool **MPEGIO2_getBoxEnable**(unsigned long chanNum,
unsigned char boxNum,
bool &enable);

Function: Get the Enable/Disable status of an existing Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Output Parameter **enable**: true for enable, false for disable
Return true for success
Note: When enabled (default) the Box Overlay appears, when disabled it disappears from the video surface but is not deleted.

MPEGIO2_API bool **MPEGIO2_enableAllOverlayBoxItems**(unsigned long chanNum, bool enable);

Function: Enable/disable all OT_BOX type overlay items (let them appear or disappear).
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number

Input Parameter **enable**: true to enable, false to disable all OT_BOX type overlay items
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_enableAllOverlayBoxItemBoundary(unsigned long chanNum,
                                                          bool enable);
```

Function: Enable/disable all OT_BOX type overlay items' boundaries.
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input Parameter **enable**: true to enable, false to disable all OT_BOX type overlay items' boundaries.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setBoxCLUT(   unsigned long chanNum,
                                        int CLUTIndex,
                                        unsigned char R,
                                        unsigned char G,
                                        unsigned char B);
```

Function: Set the CLUT(Colour Look Up Table) entry value for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input Parameter **CLUTIndex**: the CLUT entry number, between [0, 3] inclusive.
Input Parameter **R**: the Red component value of the RGB colour of the CLUT entry.
Input Parameter **G**: the Green component value of the RGB colour of the CLUT entry.
Input Parameter **B**: the Blue component value of the RGB colour of the CLUT entry.
Return true for success.
Note 1: Each channel has a 4-entry CLUT (Colour Look Up Table) for Box Overlay's colour, each entry in the CLUT can be set to any RGB colour value, and each Box overlay can use any one of these 4-entry's value as its painting colour.
Note 2: Box overlay's CLUT is totally un-related to the CLUT used by all other non-Box overlays.

```
MPEGIO2_API bool  MPEGIO2_getBoxCLUT(   unsigned long chanNum,
                                        int CLUTIndex,
                                        unsigned char &R,
                                        unsigned char &G,
                                        unsigned char &B);
```

Function: Get CLUT(Colour Look Up Table) entry values for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 channel number
Input Parameter **CLUTIndex**: the CLUT entry number, between [0, 3] inclusive.
Output Parameter **R**: the Red component value of the RGB colour of the CLUT entry.
Output Parameter **G**: the Green component value of the RGB colour of the CLUT entry.
Output Parameter **B**: the Blue component value of the RGB colour of the CLUT entry.
Return true for success
Note 1: Each channel has a 4-entry CLUT (Colour Look Up Table) for Box Overlay's colour, each entry in the CLUT can be set to any RGB colour value, and each Box overlay can use any one of these 4-entry's value as its paiting colour.
Note 2: Box overlay's CLUT is totally un-related to the CLUT used by all other non-Box overlays.

```
MPEGIO2_API bool  MPEGIO2_setBoxColour(  unsigned long chanNum,
                                         unsigned char boxNum,
                                         unsigned char colour);
```

Function: Set colour value for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Input Parameter **colour**: the new colour value, between [0, 15] inclusive.

0 White (75% Amplitude 100% Saturation) (default)
1 Yellow (75% Amplitude 100% Saturation)
2 Cyan (75 % Amplitude 100 Saturation)
3 Green (75% Amplitude 100% Saturation)
4 Magenta (75% Amplitude 100% Saturation)
5 Red (75% Amplitude 100% Saturation)
6 Blue (75% Amplitude 100% Saturation)
7 0% Black

8 100% White
9 50% Gray
10 25% Gray
11 Blue (75% Amplitude 75% Saturation)
12 Defined by Box CLUT index 0
13 Defined by Box CLUT index 1
14 Defined by Box CLUT index 2
15 Defined by Box CLUT index 3

Return true for success

MPEGIO2_API bool **MPEGIO2_getBoxColour**(unsigned long chanNum,
unsigned char boxNum,
unsigned char &colour);

Function: Get colour value for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Output Parameter **colour**: the new colour value, between [0, 15] inclusive.

0 White (75% Amplitude 100% Saturation) (default)
1 Yellow (75% Amplitude 100% Saturation)
2 Cyan (75 % Amplitude 100 Saturation)
3 Green (75% Amplitude 100% Saturation)
4 Magenta (75% Amplitude 100% Saturation)
5 Red (75% Amplitude 100% Saturation)
6 Blue (75% Amplitude 100% Saturation)
7 0% Black
8 100% White
9 50% Gray
10 25% Gray
11 Blue (75% Amplitude 75% Saturation)
12 Defined by Box CLUT index 0
13 Defined by Box CLUT index 1
14 Defined by Box CLUT index 2
15 Defined by Box CLUT index 3

Return true for success

MPEGIO2_API bool **MPEGIO2_setBoxAlpha**(unsigned long chanNum,
unsigned char boxNum,
unsigned char alpha);

Function: Set alpha value for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Input Parameter **alpha**: = 1 for 50% alpha, = 2 for 75% alpha, = 3 for 25% alpha, other values: no alpha (solid)
Return true for success
Note 1: All Box Overlay Items share the same (non-zero) alpha settings.
Note 2: Box Overlay's Alpha has no effect on Non-Box Overlay Items' alpha settings.

MPEGIO2_API bool **MPEGIO2_getBoxAlpha**(unsigned long chanNum,
unsigned char boxNum,
unsigned char &alpha);

Function: Get alpha value for Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Output Parameter **alpha**: = 1 for 50% alpha, = 2 for 75% alpha, = 3 for 25% alpha, other values: no alpha (solid)
Return true for success

MPEGIO2_API bool **MPEGIO2_setBoxPos**(unsigned long chanNum,
unsigned char boxNum,
unsigned short x,
unsigned short y,
unsigned short width,

unsigned short height);

Function: Set new position for a Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
input parameter **x**: Box's horizontal position on video frame in pixel unit, must within 0 ~ 719
input parameter **y**: Box's vertical position on video frame in pixel unit,
must within 0 ~ 575 for PAL, within 0 ~ 479 for NTSC
input parameter **width**: Box' width in pixel unit, must be within 1~720
input parameter **height**: Box' height in pixel unit, must be within 1~576 for PAL, 1~480 for NTSC
Return true for success.
Note: **x+width** must be < 720, **y+height** must be < 576 for PAL, < 480 for NTSC.

```
MPEGIO2_API bool  MPEGIO2_getBoxPos(    unsigned long chanNum,
                                        unsigned char boxNum,
                                        unsigned short &x,
                                        unsigned short &y,
                                        unsigned short &width,
                                        unsigned short &height);
```

Function: Get the position of a Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Output parameter **x**: Box's horizontal position on video frame in pixel unit, must within 0 ~ 719
Output parameter **y**: Box's vertical position on video frame in pixel unit, within 0 ~ 575 for PAL, within 0 ~ 479 for NTSC
Output parameter **width**: Box' width in pixel unit, within 1~720
Output parameter **height**: Box' height in pixel unit, within 1~576 for PAL, 1~480 for NTSC
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setBoxBoundary(unsigned long chanNum,
                                         unsigned char boxNum,
                                         bool enable);
```

Function: Enable/Disable boundary for a Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
input parameter **enable**: true will enable boundary, false will disable bounary for the Box overlay
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getBoxBoundary(unsigned long chanNum,
                                         unsigned char boxNum,
                                         bool &enable);
```

Function: Get Enable/Disable boundary status for a Box overlay on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **boxNum**: the box number, between [0, 3] inclusive.
Output parameter **enable**: true will enable boundary, false will disable bounary for the Box overlay
Return true for success.

4.13.3 Overlay Item Management

MPEGIO2_API unsigned short **MPEGIO2_totalOverlay**(unsigned long chanNum);

Function: Get the total number of existing overlay items on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return number of total overlay items created for the channel, 0 means none was created
Note: the most recently created overlay item always has its index number == total overlay number - 1.

```
MPEGIO2_API bool  MPEGIO2_deleteOverlay( unsigned long chanNum,
                                         unsigned short overlayNum,
                                         bool eraseScreen = true);
```

Function: Delete an existing overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **overlayNum**: the number of the overlay to be deleted. If -1, all overlay items will be deleted
Input parameter **eraseScreen**: if true, will erase video frame's overlay content for the deleted overlay
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_redrawOverlay(unsigned long chanNum,
                                        unsigned short overlayNum,
                                        HWND wnd);
```

Function: Redraw an existing overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **overlayNum**: the index number of the overlay item to be redrawn.
Input parameter **wnd**: window handle that will receive messgaes sent from SDK to application program
Return true for success.
Note 1: Redraw one Overlay item on the channel is an asynchronous process:
when this function returns, it does not necessarily mean the redrawing process is finished: only when a message MPEGIO2_MSG_OK_REDRAW_1OSD (or MPEGIO2_MSG_ERR_REDRAW_1OSD) is received by the caller application then the redrawing really succeeded or failed.
Note 2: If the overlayNum is a Box Overlay (type is OT_BOX) then this function simply enables its colour but does not enable its boundary(Use **MPEGIO2_setBoxBoundary** to enable Box Overlay's boundary).

```
MPEGIO2_API bool  MPEGIO2_redrawOverlayText( unsigned long chanNum,
                                             unsigned short overlayNum,
                                             HWND wnd,
                                             char *text,
                                             wchar_t *textW = 0,
                                             unsigned short *x = 0,
                                             unsigned short *y = 0,
                                             bool updateY = false,
                                             bool wrapY = false);
```

Function: Redraw an existing Text overlay item on a channel with new text string content and position.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number.
Input parameter **overlayNum**: the index number of the overlay item to be redrawn.
Input parameter **wnd**: window handle that will receive messgaes sent from SDK to application program.
Input parameter **text**: new text content as ASCII string, must be non-empty null-terminated string of < 512 bytes long.
Input parameter **textW**: If the item is a wide character string text item, this will be the new Unicode text string content.
Input parameter **x**: if not null, must point the a variable holding the new horizontal start position of the text to be redrawn.
Input parameter **y**: if not null, must point the a variable holding the new vertical start position of the text to be redrawn.
Input parameter **updateY**: If true, ignore parameter y, automatically increase vertical position by the height of the text string before redrawing the text.
Input parameter **wrapY**: If true and **updateY** is true, then when the text string's current Y value plus text string's height exceeds the video frame's bottom edge, reset y position to initial Y (as the text overlay item was first created) before redrawing the text overlay item;
If false and **updateY** is true,then when the text string's current Y value plus text string's height exceeds the video frame's bottom edge, move the overlay image from the inital Y plus height towards the current Y plus text string height upwards by one height(i.e., shift the text string area upwards one line) before redrawing the new text string at the current Y position.
Return true for success.
Note 1: Redrawing one text Overlay item on the channel is an asynchronous process:

when this function returns, it does not necessarily mean the redrawing process is finished: only when a message MPEGIO2_MSG_OK_REDRAW_1OSD (or MPEGIO2_MSG_ERR_REDRAW_1OSD) is received by the caller application then the redrawing really succeeded or failed.
Note 2: If x or y is null then the horizontal or vertical start position of the redreawn item will use current position values.
Note 3: If parameter **text** is not NULL then the new ASCII string pointed to by **text** will be used. If **text** is NULL then the Unicode string pointed to by **textW** will be used, If both **text** and **textW** have no content this function will fail.

MPEGIO2_API bool **MPEGIO2_redrawAllOverlays**(unsigned long chanNum,
HWND wnd,
bool blocking = false);

Function: Redraw all existing overlay items on an **MPEGIO2** channel regardless the Video Preview is On or Off.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **wnd**: the application supplied window all SDK messages will be sent to
Input parameter **blocking**: if true, do not return from this function until it finishes drawing all overlay items, otherwise start overlay drawing thread and return immediately: when all overlay items are drawn a message MPEGIO2_MSG_OK_REDRAW_ALLOSD (or MPEGIO2_MSG_ERR_REDRAW_ALLOSD) will be sent to "**wnd**" window indicating success or failure of drawing all overlays
Return true for success.
Note 1: if the channel has no overlay created false is returned.
Note 2: On startiong up **MPEGIO2** SDK **does not** automatically redraw existing Overlay Items if the Video Preview if not turned on(**MPEGIO2_startPreviewWindow** is never called): in this case the application can call **MPEGIO2_redrawAllOverlays** to redraw existing Overlay items while keeping the Video Preview Window not being created. If the Video Preview is turned on and **MPEGIO2_startPreviewWindow** get value "true" for its parameter "**redrawOverlays**", the SDK will automatically redraw overlays so the Application software does not need to call this function to redraw overlays. Redrawing all overlays while keeping the Video Preview Window not being created is useful for silently recording/streaming MPEG video together with overlay items.

MPEGIO2_API bool **MPEGIO2_clearOverlay**(unsigned long chanNum,
unsigned long x,
unsigned long y,
unsigned long width,
unsigned long height);

Function: Clear all overlay pixels within an area on video frame of a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **x/y/width/height**: specify the overlay area x/y/width/height on video frame in pixel unit.
width must be within [0, 720], **height** must be within [0, 576],
if **width** is 0 then the cleared area's width will be from x to 720(The full width of the OSD display area)
if **height** is 0 then the cleared area's height will be from y to 576(The full height of the OSD display area)
Note: This function does not clear the OT_BOX type overlay items from the screen, OT_TIMER type overlay will be cleared but will reappear in 1 sec.

MPEGIO2_API bool **MPEGIO2_eraseOverlay**(unsigned long chanNum, unsigned short overlayNum);
Function: Erase the display of an existing overlay item on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **overlayNum**: the index number of the overlay item to be erased.
Return true for success
Note: This function only erase the overlay item's display but does not delete the item.

MPEGIO2_API bool **MPEGIO2_moveOverlayPos**(unsigned long chanNum,
unsigned short overlayNum,
unsigned short newX,
unsigned short newY);

Function: Move an existing overlay item on the video frame to a new position for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input Parameter **overlayNum**: the index number of the existing overlay item
Input parameter **newX**: new horizontal position on video frame for the overlay
Input parameter **newY**: new vertical position on video frame for the overlay
Return true if succeed.

MPEGIO2_API bool **MPEGIO2_disableOverlay**(unsigned long chanNum,
bool disable,
bool redrawAllOSD = true);

Function: Temporarily disable(or enable) all overlays for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **disable**: true to temporarily disable (but not delete) all overlays
Return true if succeed
Note 1: By default all overlays defined are enabled, this function is only called when a temparory overlay disable is needed
Note 2: If no overlay was created for this channel this function returns false
Note 3: calling this function with disable = true when the channel's Overlay has been disabled, or
calling this function with disable = false when the channel's Overlay has been enabled,
will result in no effect and returning false.

MPEGIO2_API bool **MPEGIO2_isOverlayDisabled**(unsigned long chanNum, bool &disable);

Function: Return overlay disable status as set by function **MPEGIO2_disableOverlay** for a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **disable**: true to temporarily disable (but not delete) all overlays
Return true if succeed.

4.13.4 Download Font for Text Overlay

Although the downloading process takes time, in particular for large size fonts, using downloaded font can display text overlay instantly on video since fonts are downloaded to OSD IC's RAM. Only the ASCII values 0x20 ~ 0x7F are used as downloaded font characters, Unicode fonts are not supported.

MPEGIO2_API bool **MPEGIO2_downloadFont**(
unsigned long chanNum,
COLORREF fcolour,
COLORREF bcolour,
int fontWidth,
int fontHeight,
int fontWeight,
unsigned char italic,
char *typeFace,
unsigned short *fontNum);

Function: Download a text font on a channel to the OSD IC's RAM.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fcolour**: foreground colour in RBG format
Input parameter **bcolour**: background colour in RBG format
Input parameter **width**: font width, default is 24, same meaning as in LOGFONT.lfWidth of the Windows SDK
Input parameter **height**: font height, default is 24, same meaning as in LOGFONT.lfHeight of the Windows SDK
Input parameter **weight**: The weight of the font in the range 0 through 1000, same meaning as in LOGFONT.lfWeight of the Windows SDK
Input parameter **italic**: font italic, default is 0 (no italic), non-zero means italic
Input parameter **typeFace**: type face of the text string: The length of this string must not exceed 32 TCHAR values, including the terminating NULL.
Output parameter **fontNum**: If not NULL and function returns true, this receives the newly downloaded font's number (will be total download fonts – 1)in the downloaded font list (the 1st downloaded font has 0 as its number) that can be used later in functions "**MPEGIO2_delADLFont**", "**MPEGIO2_getADLFont**" etc.
Return: True for success, failure usually include out of RAM on the OSD IC.
Note: When OSD IC RAM is full, this function will automatically delete those downloaded fonts that no Text Overlay Item is currently using to retrieve RAM on the OSD IC. If after deleting those fonts chip RAM is still not enough for the new downloaded font this function fails. Application software can call **MPEGIO2_delADLFont** to delete fonts.

MPEGIO2_API unsigned short **MPEGIO2_listDownloadedFonts**(unsigned long chanNum,
HWND wnd);

Function: List all dowloaded fonts on a channel on the surface of the Video Preview Window for that channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **wnd**: handle of the window on which prompt message dialog will appear:

If NULL or illegal handle, the "HWND parentWnd" passed to **MPEGIO2_initSDK**() will be used.
If that is also NULL or illegal, the desktop window's handle is used.
On success, return total number of downloaded fonts for this channel, otherwise returns 0.
Note: If the channel is not currently previewing video this function has no meaning since the listed font cannot be seen.

```
MPEGIO2_API bool MPEGIO2_getADLFont( unsigned long chanNum,
                                     unsigned short fontNum,
                                     COLORREF &colour,
                                     COLORREF &bcolour,
                                     int &fontWidth,
                                     int &fontHeight,
                                     int &fontWeight,
                                     unsigned char &fontItalic,
                                     char *typeFace = 0);
```

Function: Get the characterictics of a downloaded font on a channel.
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fontNum**: the index number of the downloaded font returned by **fontNum** in **MPEGIO2_downloadFont**
Output parameter **colour**: the downloaded font colour
Output parameter **bcolour**: the downloaded font background colour
Output parameter **fontWidth**: the font width
Output parameter **fontHeight**: the font Height
Output parameter **fontWeight**: the font Weight
Output parameter **fontItalic**: the font Italic, non zero means italic font.
Output parameter **typaFace**: the font typeface, caller must supply a character string with >= 32 bytes long.
If this is NULL then no typeface will be returned.
On success, return true.

```
MPEGIO2_API unsigned short MPEGIO2_getDLFontNums(unsigned long chanNum);
```

Function: Return the total number of all downloaded fonts on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Return the number of total downloaded fonts, if 0 is returned there is no downloaded font for this channel.

```
MPEGIO2_API bool MPEGIO2_delADLFont(unsigned long chanNum, unsigned short fontNum);
```

Function: Delete a downloaded font on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **fontNum**: the index number of the downloaded font, between 0 and **MPEGIO2_getDLFontNums**()-1
On success, return true.

```
MPEGIO2_API unsigned short MPEGIO2_getTextDLFontNum(
                                     unsigned long chanNum,
                                     unsigned short textOverlayNum,
                                     bool *downloaded);
```

Function: Test if a Text Overlay Item uses downloaded font, if yes, return the downloaded font number it uses.
Input parameter **chanNum**: 0 ~ totalChans-1, the **MPEGIO2** channel number
Input parameter **textOverlayNum**: the Overlay Item Number
Output parameter **downloaded**: If not NULL and the function returns non 0xFFFF, this parameter will indicate
if the downloaded font the overlay item "textOverlayNum" uses has been downloaded into OSD IC's RAM or not.
Return 0xFFFF (-1) if the overlay item is not text overlay, or it does not use downloaded font,
otherwise return the index number of the downloaded font used by Text Overlay Item "textOverlayNum",
which will be less than the total number of all downloaded fonts returned by **MPEGIO2_getDLFontNums**().

4.14 Digital I/O Functions

Each **MPEGIO2** channel has 4 **Digital I/O Pins** accessible from the **Breakout Box**'s Top Layer: Each pin can be configured as either Input (default) or Output direction. Input direction Digital I/O pins can accept external High/Low signals and notify the application software through interrupt notification callback function, or the application software can poll the status of these pins using SDK functions. Output direction digital I/O pins can be set to High or Low by the application software using SDK functions. A ground pin is also available on the **Breakout Box** for connection to the external device. The **MPEGIO2** SDK also provides Video Input Status Change functions for the first 4 Video Input Sources (VIN0 ~ VIN3) when their Video Loss, Motion Detect, Blind Detect or Night Detect happens.

4.14.1 Digital I/O Pin Functions

MPEGIO2_API bool **MPEGIO2_DIOSetDirection**(ULONG chanNum, BYTE pins, BYTE vals);

Function: Set Digital IO Directions (each DIO pin can be set as either Input or Ouput: Power-up default is Input)
Input Parameter **chanNum**: the channel number, must be between 0 ~ totalChans-1
Input Parameter **pins**: the lowest 4 bits indicate which of the 4 DIO pins will be set their Input/Output directions:
1 means that corresponding pin will be set I/O direction using the corresponding bit value in parameter "**vals**",
0 means that corresponding pin will not be affected in this function call.
If the lowest 4 bits of this parameter are all zeroes this function returns false and does nothing.
The values of the higher 4 bits of this parameter have no effect in this function.
Input Parameter **vals**: the lowest 4 bits indicate Digital IO directions to be set:
1 means Input Direction, 0 means Output direction.
Note only bits indicated as "1" by parameter "**pins**" in its corresponding bits will be set IO direction,
bits indicated as "0" by parameter "**pins**" in its corresponding bits will not be set direction regardless value in "**vals**"
Return: true for success

MPEGIO2_API bool **MPEGIO2_DIOGetDirection**(ULONG chanNum, BYTE &directionVals);

Function: Retrieve the current 4 Digital IO Directions (each DIO bit can be input or ouput direction)
Input Parameter **chanNum**: the channel number, must be between 0 ~ totalChans-1
Output Parameter **directionVals**: On a successful call, the lowest 4 bits will receive the current Digital IO pin directions:
1 means Input Direction, 0 means Output direction.
Return: true for success
Note: After a power reset all 4 DIO pins are in Input Direction, but if an **initFile** MPEGIO2.ini exists in the same folder as the application program, calling **MPEGIO2_initSDK**() will set the DIO directions according to the values remembered in MPEGIO2.ini which indicate the previously set DIO directions by the application program.

MPEGIO2_API bool **MPEGIO2_DIOSetValue**(
ULONG chanNum,
BYTE pins,
BYTE vals,
bool restoreDirection = true);

Function: Set Digital IO Pin Output Value
Input Parameter **chanNum**: the channel number, must be between 0 ~ totalChans-1
Input Parameter **pins**: the lowest 4 bits indicate which Digital IO pins to set values:
"1" means to set that pin's output value, "0" means do not set that pin's value.
If this parameter's lowest 4 bits are all zeroes then this function returns false and does nothing.
Input Parameter **vals**: the lowest 4 bits indicate Digital IO pin output values to be set:
only pins with their corresponding values in parameter "**pins**" indicated as "1" will be set.
When a bit in "**vals**" is 1 the output voltage at this pin will be aprrox. +5 Volt,
When a bit in "**vals**" is 0 the output voltage at this pin will be 0 Volt.
Input Parameter **restoreDirection**: If true, this fucntion will restore all DIO pins direction settings on exit,
if false, the DIO pins' directions will remain as set by this function. See "Note" below.
Return: true for success
Note: This function will automatically set those pins indicated as "1" in parameter "**pins**"
as Output Direction before outputting their values,
and if parameter "**restoreDirection**" is true, will restore their direction settings after outputting pin values.

MPEGIO2_API bool **MPEGIO2_DIOGetValue**(ULONG chanNum, BYTE &vals);
Function: Read Digital IO Pin Input Value
Input Parameter **chanNum**: the **MPEGIO2** channel number, must be between 0 ~ totalChans-1
Output Parameter **vals**: the lowest 4 bits indicate the 4 Digital IO pin values read:
bit 0 for pin 0's value, bit 1 for pin 1's value, bit 2 for pin 2's value, bit 3 for pin 3's value.
Input DC voltage -0.5V ~ +0.3V will be recognized as Low(0), +0.7V ~ +5.5V will be recognized as High(1).
Return: true for success
Note: This function will read all 4 DIO pins' current input values which indicate these pins' current incoming logic levels regardless their Input/Output Direction settings.

```
MPEGIO2_API bool  MPEGIO2_setDIONotify(  unsigned long chanNum,
                                         bool enable,
                                         bool usePolling = true,
                                         HWND wnd = 0,
                                         MPEGIO2_callback_INTC callbackFuncC = 0,
                                         MPEGIO2_callback_INTS callbackFuncS = 0,
                                         unsigned long responseTime = 500);
```

Function: Enable/disable SDK to notify the user application for Digital IO Input Pins 0 ~ 3 on an **MPEGIO2** channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **enable**: true to enable, false to disable the notification for digital input value change.
All the following parameters will have no meanings if parameter "**enable**" is false.
Input parameter **usePolling**: False means DIO Input value changes will trigger interrupt at a GPIO pin of the Video preview IC,
True (default) means SDK will use polling to find out if DIO Input has value changes
Input parameter **wnd**: If this is a valid window handle then when an enabled interrupt happens the SDK will send message MPEGIO2_MSG_WM_DIGITAL_INPUT_INTERRUPT to this window.

Input parameter **callbackFuncC**: If this is not NULL (must point to a valid function of type **MPEGIO2_callback_INTC**) then when an enabled interrupt happens the **MPEGIO2** SDK will call this function.
See "Note 2" below for this function's details.
Input parameter **callbackFuncS**: If this is not NULL (must point to a valid function of type **MPEGIO2_callback_INTS**) then when an enabled interrupt happens the **MPEGIO2** SDK will call this function
See "Note 2" below for this function's details.
Input parameter **responseTime**: When the enabled interrupt happends, the maximum time in milli-seconds the **MPEGIO2** SDK could wait before notifying the application through sending message to **wnd** and / or calling functions **callbackFuncC/callbackFuncS**, minimum is 10.
Return: true for success.

Note 1: Those Digital IO Pins to have their input change notification enabled must be set up as "Input Direction" through function **MPEGIO2_DIOSetDirection**(ULONG chanNum, BYTE pins, BYTE vals) otherwise there will be no notifications on those pins.
Note 2: **MPEGIO2.h** definitions for call back function type **MPEGIO2_callback_INTC** and **MPEGIO2_callback_INTS**:

```
typedef void (_cdecl    * MPEGIO2_callback_INTC)
        (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
typedef void (__stdcall * MPEGIO2_callback_INTS)
        (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
```

parameter **chanNum**: the **MPEGIO2** channel number on which a Video Input or Digital Input Interrupt occured.
parameter **type**: 0 for Video Input (Video Source 0 ~ 3) caused interrupt,
1 for Digital Input Socket (0 ~ 3) caused interrupt
parameter **num**: For type==0, each of its bits[7:4] bits indicate which input source (0 ~ 3) caused the interrupt:
could be multiple sources (more than one bits could be set to 1: more than one among VIN0~VIN3).
The value of bits[3:0] indicate which kind of interrupt occured:
Bit0 == 1 to indicate video loss interrupt occured,
Bit1 == 1 to indicate motion detect interrupt occured,
Bit2 == 1 to indicate indicate blind detect interrupt occured,
Bit3 == 1 to indicate night detect interrupt occured.
Note more than one bits in Bits[3:0] could be set to 1 to indicate multiple kinds of interrupts occured.

For type==1, the bits 0~3 indicate the current input ports' values (only valid for those ports configured as input direction)
the bits 4~7 indicate the previous input ports' values (only valid for those ports configured as input

direction)
It is the difference between the current and previous input ports' values that caused interrupt.
parameter **MDPtr**: When parameter type is 0 (Video Input Interrupt Occured):
this represents a 24X4 bytes array, each 24 bytes represent 12 row X 16 column cells' motion detect status:
1 means that cell has motion detected, 0 means that cell has no motion detected.
When parameter type is 1 (Digital Input Interrupt occured): this parametr is ignored.

```
MPEGIO2_API bool  MPEGIO2_getDIONotify(  unsigned long chanNum,
                                         bool &enable,
                                         bool *usePolling = 0,
                                         HWND *wnd = 0,
                                         MPEGIO2_callback_INTC *callbackFuncC = 0,
                                         MPEGIO2_callback_INTS *callbackFuncS = 0,
                                         unsigned long *responseTime = 0);
```

Function: Retrieve value change notification settings for Digital IO Input Pins 0 ~ 3 on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **enable**: get either true to enable, or false to disable the input status change notification for digital input.
Output parameter **usePolling**: will receive the **usePolling** value as set by calling **MPEGIO2_setDIONotify**.
Output parameter **wnd**: If not NULL, window handle to receive message
MPEGIO2_MSG_WM_DIGITAL_INPUT_INTERRUPT from SDK.
Output parameter **callbackFuncC**: If not NULL will receive a pointer to a valid function of type
MPEGIO2_callback_INTC or NULL
Output parameter **callbackFuncS**: If not NULL will receive a pointer to a valid function of type
MPEGIO2_callback_INTS or NULL
Output parameter **responseTime**: If not NULL, will receive the responseTime value as set by calling
MPEGIO2_setDIONotify, default is 500ms
Return: true for success

4.14.2 Video Source Status Functions

```
MPEGIO2_API bool  MPEGIO2_setVideoInputNotify(
                                         unsigned long chanNum,
                                         ULONG videoSrc,
                                         ULONG interrupts,
                                         HWND wnd,
                                         MPEGIO2_callback_INTC callbackFuncC = 0,
                                         MPEGIO2_callback_INTS callbackFuncS = 0,
                                         unsigned char *MDPtr = 0,
                                         bool copyMDPtr = false,
                                         unsigned long responseTime = 500);
```

Function: Enable/disable notification for input status changes for Video Input Sources VIN0 ~ VIN3 on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number

Input parameter **videoSrc**: Setting each bit in the lowest 4 bits (Bit0, Bit1, Bit2, Bit3) to 1 will indicate
to enable status change notification on Video Source 1, 2, 3, 4 (VIN0~VIN3) respectively.
Setting each individual bit to 0 will disable that Video Source's status change notification.

Input parameter **interrupts**: Setting 1 to individual bits in Bits[3:0] indicates to enable/disable these status notifications:
bit0 = Video Loss interrupt,
bit1 = Motion Detect interrupt,
bit2 = Blind Detect interrupt,
bit3 = Night Detect interrupt.
Setting 0 to all bits in Bits[3:0] indicates to disable all interrupt notification.

Input parameter **wnd**: If this is a valid window handle then when an enabled interrupt happens the **MPEGIO2** SDK
will send message MPEGIO2_MSG_WM_VIDEO_INPUT_INTERRUPT to this window.
Input parameter **callbackFuncC**: If this is not NULL (must point to a valid function of **type MPEGIO2_callback_INTC**)
then when an enabled interrupt happens the **MPEGIO2** SDK will call this function.
See Note below for this function’s details.

Input parameter **callbackFuncS**: If this is not NULL (must point to a valid function of type **MPEGIO2_callback_INTS**) then when an enabled interrupt happens the **MPEGIO2** SDK will call this function. See Note below for this function's details.

Input parameter **MDPtr**: This is only meaningful when bit1 of **interrupts** is 1 (Motion Detect happens):
If not NULL, must point to a >= 24 X 4 bytes array, which represnts the Motion Detect Mask Values for the 4 Video Input Sources (VIN0~VIN3): bits in each 24 bytes represent the 16X12=192 Mask Cells:
bit value 1 means the mask cell is masked (no motion detect will happen on that cell),
bit value 0 means the mask cell is not masked (motion detect will happen on that cell),
By default all cells in all video sources are not masked: motion detect will happen on all cells.
Video frame in each video source is devided into 12 row X 16 column cells each of the cells can be masked or unmasked (default) for detecting motion (unmasked) or not detecting motion (masked).
When this parameter is not null and parameter **copyMDPtr** is true, each cell will contain the motion detect status when motion detect happens on that video input source.
In each 24 bytes, byte 0 represent cell row 0 column 0~7, byte1 represent cell row 0 column 8~15
byte 2 represent cell row 1 column 0~7, byte3 represent cell row 1 column 8~15
byte 4 represent cell row 2 column 0~7, byte5 represent cell row 2 column 8~15
... ...
byte 22 represent cell row 11 column 0~7, byte23 represent cell row 11 column 8~15

Input parameter **copyMDPtr**: If true and parameter **MDPtr** is pointing to a 24X4 byte array, then each time motion detect interrupt happens, the 4X192 cells' motion detect status of all video input sources will be copied to array **MDPtr**: each bit with value 1 means that cell has motion detected, each bit of 0 means that cell has no motion detected.

Input parameter **responseTime**: When the enabled interrupt happends, the maximum time in milli-seconds the **MPEGIO2** SDK could wait before notifying the application through sending message to **wnd** and / or calling functions **callbackFuncC/callbackFuncS**, minimum is 10.

Note: **MPEGIO2.h** definitions for call back function type **MPEGIO2_callback_INTC** and **MPEGIO2_callback_INTS**:

```
typedef void (_cdecl   * MPEGIO2_callback_INTC)
        (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
typedef void (__stdcall * MPEGIO2_callback_INTS)
        (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
```

parameter **chanNum**: the **MPEGIO2** channel number on which a Video Input or Digital Input Interrupt occured.
parameter **type**: 0 for Video Input (Video Source 0 ~ 3) caused interrupt,
1 for Digital Input Socket (0 ~ 3) caused interrupt
parameter **num**: For type==0, each of its bits[7:4] bits indicate which input source (0 ~ 3) caused the interrupt: could be multiple sources (more than one bits could be set to 1: more than one among VIN0~VIN3).
The value of bits[3:0] indicate which kind of interrupt occured:
Bit0 == 1 to indicate video loss interrupt occured,
Bit1 == 1 to indicate motion detect interrupt occured,
Bit2 == 1 to indicate indicate blind detect interrupt occured,
Bit3 == 1 to indicate night detect interrupt occured.
Note more than one bits in Bits[3:0] could be set to 1 to indicate multiple kinds of interrupts occured.
For type==1, the bits 0~3 indicate the current input ports' values (only valid for those ports configured as input direction)
the bits 4~7 indicate the previous input ports' values (only valid for those ports configured as input direction)
It is the difference between the current and previous input ports' values that caused interrupt.
parameter **MDPtr**: When parameter type is 0 (Video Input Interrupt Occured):
this represents a 24X4 bytes array, each 24 bytes represent 12 row X 16 column cells' motion detect status:
1 means that cell has motion detected, 0 means that cell has no motion detected.
When parameter type is 1 (Digital Input Interrupt occured): this parametr is ignored.

MPEGIO2_API bool **MPEGIO2_getdVideoInputNotify**(unsigned long chanNum, unsigned long &videoSrc);

Function: Get the interrupt status reporting setup for video input sources 0 ~ 3 on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Output parameter **videoSrc**: Indicate which Input Video Source will report Interrupt: by setting a bit to 1 in the lowest 16 bits:
bit0 ~ bit3 indicate Video Source 1, 2, 3, 4 will report Video Loss interrupt, respectively;
bit4 ~ bit7 indicate Video Source 1, 2, 3, 4 will report Motion Detect interrupt, respectively;
bit8 ~ bit11 indicate Video Source 1, 2, 3, 4 will report Blind Detect interrupt occured, respectively;
bit12~ bit15 indicate Video Source 1, 2, 3, 4 will report Night Detect interrupt occured, respectively;

Return true for success.
Note: calling this function will always receive the current setup status of all Video Input Interrut Reporting, regardless if function **MPEGIO2_setVideoInputNotify** was ever called.

MPEGIO2_API bool **MPEGIO2_getdVideoInputNotifyFuncs**(
unsigned long chanNum,
MPEGIO2_callback_INTC *callbackFuncC,
MPEGIO2_callback_INTS *callbackFuncS);

Function: Get the interrupt status reporting callback functions as set up by function **MPEGIO2_setVideoInputNotify**
Input parameter **chanNum**: must be between 0 ~ totalChans-1: the channel number
Output parameter **callbackFuncC**: if not NULL, will receive the **MPEGIO2_callback_INTC** type callback function as passed to function **MPEGIO2_setVideoInputNotify** (default is null)
Output parameter **callbackFuncS**: if not NULL, will receive the **MPEGIO2_callback_INTS** type callback function as passed to function **MPEGIO2_setVideoInputNotify** (default is null)
Return true for success.
Note: calling this function will always receive the current Video Input Interrut Status Reporting callback function values, regardless if function **MPEGIO2_setVideoInputNotify** was ever called.

MPEGIO2_API bool **MPEGIO2_readVideoInterrupt**(unsigned long chanNum,
unsigned long interrupts,
unsigned long &videoSrc);

Function: Read the interrupt status for video input sources 0 ~ 3 on a channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **interrupts**: Set 1 to individual bits in Bits[3:0] to indicate to read these interrupt status:
bit0 = Read Video Loss interrupt Status,
bit1 = Read Motion Detect interrupt Status,
bit2 = Read Blind Detect interrupt Status,
bit3 = Read Night Detect interrupt Status.
If the lowest 4 bits are all zeroes this function does nothing and returns false.
Output parameter **videoSrc**: Receive which Input Video Source Has Interrupt Occured:
by setting a bit to 1 in the lowest 16 bits:
bit0 ~ bit3 indicate Video Source 1, 2, 3, 4 has Video Loss interrupt occured, respectively;
bit4 ~ bit7 indicate Video Source 1, 2, 3, 4 has Motion Detect interrupt occured, respectively;
bit8 ~ bit11 indicate Video Source 1, 2, 3, 4 has Blind Detect interrupt occured, respectively;
bit12~ bit15 indicate Video Source 1, 2, 3, 4 has Night Detect interrupt occured, respectively;
Return true for success.
Note: this function is only needed when application wishes to manually check (poll) the video input interrupt status. Since calling function **MPEGIO2_setVideoInputNotify** can usually enable automatic reporting of video input interrupt by supplying the parameters **wnd**, **callbackFuncC**, and **callbackFuncS**, this function is not really needed, unless function **MPEGIO2_setVideoInputNotify** was never called, or called with the parameters **wnd**, **callbackFuncC**, and **callbackFuncS** not supplied.

MPEGIO2_API bool **MPEGIO2_setBDCTS**(unsigned long chanNum,
unsigned short inputSrc,
unsigned char sen);

Function: Set Blind Detect Cell Threshhold Sensitivity on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5^{th} input source has no Blind Detect
Input parameter **sen**: sensitivity value, 0 ~ 3: smaller value means lower threshhold: more sensitive
Return true for success.

MPEGIO2_API bool **MPEGIO2_getBDCTS**(unsigned long chanNum,
unsigned short inputSrc,
unsigned char &sen);

Function: Get Blind Detect Cell Threshhold Sensitivity on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5^{th} input source has no Blind Detect
Output parameter **sen**: sensitivity value, 0 ~ 3: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setBDLVS(unsigned long chanNum,
                                  unsigned short inputSrc,
                                  unsigned char sen);
```
Function: Set Blind Detect Level Threshhold Sensitivity on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Blind Detect
Input parameter **sen**: sensitivity value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getBDLVS(unsigned long chanNum,
                                  unsigned short inputSrc,
                                  unsigned char &sen);
```
Function: Get Blind Detect Level Threshhold Sensitivity on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has Blind Detect
Output parameter **sen**: sensitivity value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setNDTMTS(unsigned long chanNum,
                                   unsigned short inputSrc,
                                   unsigned char sen);
```
Function: Set Night Detect threshold of temporal Sensitivity on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Night Detect
Input parameter **sen**: sensitivity value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getNDTMTS(unsigned long chanNum,
                                   unsigned short inputSrc,
                                   unsigned char &sen);
```
Function: Get Night Detect threshold of temporal Sensitivity on a Video Input Source for a Channel
Input parameter chanNum: 0 ~ totalChans-1: the channel number
Input parameter inputSrc: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Night Detect
Output parameter sen: sensitivity value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setNDLVTS(unsigned long chanNum,
                                   unsigned short inputSrc,
                                   unsigned char sen);
```
Function: Set the threshold of level for night detection on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Night Detect
Input parameter **sen**: threshold value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getNDLVTS(unsigned long chanNum,
                                   unsigned short inputSrc,
                                   unsigned char &sen);
```
Function: Get the threshold of level for night detection on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Night Detect
Output parameter **sen**: threshold value, 0 ~ 15: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setMDLVSE(unsigned long chanNum,
                                   unsigned short inputSrc,
                                   unsigned char sen);
```
Function: Set the level sensitivity of motion detect on a Video Input Source for a Channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Input parameter **sen**: threshold value, 0 ~ 31: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getMDLVSE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char &sen);
```

Function: Get the level sensitivity of motion detect on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Output parameter **sen**: threshold value, 0 ~ 31: smaller value means lower threshhold: more sensitive
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setMDCESE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char sen) ;
```

Function: Set the threshold of sub-cell number for motion detection on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Input parameter **sen**: threshold value, 0 ~ 3:

0 Motion is detected if 1 sub-cell has motion (More sensitive) (default)
1 Motion is detected if 2 sub-cells have motion
2 Motion is detected if 3 sub-cells have motion
3 Motion is detected if 4 sub-cells have motion (Less sensitive)

Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getMDCESE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char &sen);
```

Function: Get the threshold of sub-cell number for motion detection on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Output parameter **sen**: threshold value, 0 ~ 3:

0 Motion is detected if 1 sub-cell has motion (More sensitive) (default)
1 Motion is detected if 2 sub-cells have motion
2 Motion is detected if 3 sub-cells have motion
3 Motion is detected if 4 sub-cells have motion (Less sensitive)

Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setMDSPED(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char speed);
```

Function: Set the velocity of motion detector on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Input parameter **speed**: must be within 0 ~ 62 to Control the velocity of motion detector.

Large value is suitable for slow motion detection.
0 - 1 field intervals (default)
1 - 2 field intervals
... ...
61 - 62 field intervals
62 - 63 field intervals

Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getMDSPED(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char &speed);
```

Function: Get the velocity of motion detector on a Video Input Source for a Channel

Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Output parameter **speed**: within 0 ~ 62 to Control the velocity of motion detector.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setMDSPSE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char sen);
```

Function: Set the spatial sensitivity of motion detector on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Input parameter **sen**: must be within 0 ~ 15 to Control the spatial sensitivity of motion detector.
0 = More Sensitive (default), ..., 15 = Less Sensitive.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getMDSPSE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char &sen);
```

Function: Get the spatial sensitivity of motion detector on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Output parameter **sen**: will be within 0 ~ 15 to Control the spatial sensitivity of motion detector.
0 = More Sensitive (default), ..., 15 = Less Sensitive.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_setMDTMSE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char sen);
```

Function: Set the temporal sensitivity of motion detector on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Input parameter **sen**: must be within 0 ~ 15 to Control the temporal sensitivity of motion detector.
0 = More Sensitive (default), ..., 15 = Less Sensitive.
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_getMDTMSE(unsigned long chanNum,
                                    unsigned short inputSrc,
                                    unsigned char &sen);
```

Function: Get the temporal sensitivity of motion detector on a Video Input Source for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **inputSrc**: 1~4 indicating one of the first 4 video input sources, the 5th input source has no Motion Detect
Output parameter **sen**: will be within 0 ~ 15 to Control the temporal sensitivity of motion detector.
0 = More Sensitive (default), ..., 15 = Less Sensitive.
Return true for success.

4.15 "initFile" Functions

MPEGIO2 SDK uses an "**initFile**" such as "**MPEGIO2.ini**" as the initialization file (see the parameter "**useInitFile**" in function **MPEGIO2_initSDK**) to save to and read from many variables' initial values that will be used throughout the SDK operation. To write and read its own variables' values(including Unicode strings), the application can also use this "**initFile**" with the functions described in this section.

The "**initFile**" has a "System" section, plus "totalChans" number of "Chan+chanNum" sections: e.g., Chan0, Chan1, etc. to save variables for Channel 0, Channel 1, respectively.

The **initFile** defaults to "**MPEGIO2.ini**" under the application program's folder, but if the application program called **MPEGIO2_setSoftwareName**() function to change the default program name then the **initFile** will be **program-name.ini**, still under the application program's folder.

The SDK reads the values of the "**initFile**" when function **MPEGIO2_initSDK** is called with parameter **useInitFile** as true(default), and always writes all parameters' values to "**initFile**" when function **MPEGIO2_endSDK** is called**.**

All functions in this section, except functions "**MPEGIO2_readVarValues"** and **"MPEGIO2_saveVarValues"**, can be called without first calling **MPEGIO2_initSDK** or **MPEGIO2_endSDK**.

MPEGIO2_API bool **MPEGIO2_writeInitString**(ULONG chanNum, char *name, char *str);
Function: Write a name=string pair value into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **str**: the string part of the pair to be written into the ini file,
this could be a NULL string or string containing NULL content in which case the written content will be "**name=**", i.e., the "**name**" will have NULL value in the written pair
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitStringW**(ULONG chanNum, char *name, wchar_t *str);
Function: Write a name=Unicode-string pair value into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be an ASCII string with some content
Input parameter **str**: the Unicode string part of the pair to be written into the ini file,
this could be a NULL string or Unicode string containing NULL content in which case the written content will be "**name=**", i.e., the "**name**" will have NULL value in the written pair
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitWORD**(ULONG chanNum,
char *name,
unsigned short val);
Function: Write a name=16-bit unsigned integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 16-bit unsigned integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitSHORT**(ULONG chanNum, char *name, short val);
Function: Write a name=16-bit signed integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,

if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 16-bit signed integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitUINT**(ULONG chanNum, char *name, unsigned int val);
Function: Write a name=32-bit unsigned integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 32-bit unsigned integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitINT**(ULONG chanNum, char *name, int val);
Function: Write a name=signed 32-bit integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 32-bit signed integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitUINT64**(ULONG chanNum,
char *name,
unsigned __int64 val);
Function: Write a name=64-bit unsigned integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 64-bit unsigned integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_writeInitINT64**(ULONG chanNum, char *name, __int64 val);
Function: Write a name=signed 64-bit integer value pair into ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be written into the ini file, must be a string with some content
Input parameter **val**: a 64-bit signed integer value to be written with the "**name**" string as a pair of the form **name=val**
Return true for success.

MPEGIO2_API bool **MPEGIO2_readInitString**(ULONG chanNum,
char *name,
char *str,
unsigned long len,
char *defaultStr);
Function: Read a name=string pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **str**: the read-in string part of the pair to be read from the ini file, this must point to an allocated string's address with enough space to hold the expected read-in bytes, otherwise a crash will happen!
Input parameter **len**: the maximum memory length in bytes unit in the supplied memory pointer "**str**", cannot be zero
Input parameter **defaultStr**: The default value to be assigned to output parameter "**str**" if the "**name**=" entry was not found in the ini file, can be NULL.
Return true for success.

```
MPEGIO2_API bool MPEGIO2_readInitStringW(ULONG chanNum,
                                         char *name,
                                         wchar_t *str,
                                         unsigned long len,
                                         wchar_t *defaultStr);
```

Function: Read a name=Unicode-string pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be an ASCII string with some content
Output parameter **str**: the read-in Unicode-string part of the pair to be read from the ini file, this must point to an allocated Unicode string's address with enough space to hold the expected read-in Unicode string, otherwise a crash will happen!
Input parameter **len**: the maximum number of Unicode characters in the supplied memory pointer "**str**", cannot be zero
Input parameter **defaultStr**: The default Unicode value to be assigned to output parameter "**str**" if the "**name=**" entry was not found in the ini file, can be NULL.
Return true for success.

```
MPEGIO2_API bool MPEGIO2_readInitWORD(ULONG chanNum,
                                      char *name,
                                      unsigned short &val,
                                      unsigned short defaultVal);
```

Function: Read a name=unsigned 16-bit-integer pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: read-in value of the pair to be read from the ini file, must point to a 16-bit unsigned integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

```
MPEGIO2_API bool MPEGIO2_readInitSHORT(ULONG chanNum,
                                       char *name,
                                       short &val,
                                       short defaultVal);
```

Function: Read a name=signed 16-bit-integer pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: the read-in value of the pair to be read from the ini file, must point to a signed 16-bit integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

```
MPEGIO2_API bool MPEGIO2_readInitUINT(  ULONG chanNum,
                                        char *name,
                                        unsigned int &val,
                                        unsigned int defaultVal);
```

Function: Read a name=unsigned 32-integer pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System".
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: read-in value of the pair to be read from the ini file, must point to an unsigned 32-bit integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

MPEGIO2_API bool **MPEGIO2_readInitINT**(ULONG chanNum,
char *name,
int &val,
int defaultVal);

Function: Read a name=signed 32-bit-integer pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: the read-in value of the pair to be read from the ini file, must point to a signed 32-bit integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

MPEGIO2_API bool **MPEGIO2_readInitUINT64**(ULONG chanNum,
char *name,
unsigned __int64 &val,
unsigned __int64 defaultVal);

Function: Read a name=unsigned 64-bit-integer pair value from ini file under a section
Input parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System"
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: the read-in value of the pair to be read from the ini file,
this must point to an unsigned 64-bit integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

MPEGIO2_API bool **MPEGIO2_readInitINT64**(ULONG chanNum,
char *name,
__int64 &val,
__int64 defaultVal);

Function: Read a name=signed 64-bit-integer pair value from ini file under a section
Iput parameter **chanNum**: the channel number,
if this is between 0 ~ totalChans-1 then the section name used will be string "Chan+chanNum",
if this is -1 then the section name used will be string "System".
Input parameter **name**: the name part of the pair to be read from the ini file, must be a string with some content
Output parameter **val**: the read-in value of the pair to be read from the ini file, must point to a signed 64-bit integer variable.
Input parameter **defaultVal**: The default value to be assigned to output parameter "**val**" if the "**name=**" entry was not found in the ini file.
Return true for success.

MPEGIO2_API bool **MPEGIO2_readVarValues**(unsigned long chanNum);

Function: Read all user-configurable variable values from the **initFile** (e.g. **MPEGIO2.ini**) under the folder **MPEGIO2** SDK resides(if the **initFile** exists)
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return: True for success
Note 1: Calling this function will make all current overlays and MPEG encoding parameters, system setup parameters etc. to restore back to previously saved values in the **initFile.**
Note 2: This function must be called after **MPEGIO2_initSDK** has been called.

MPEGIO2_API bool **MPEGIO2_saveVarValues**(unsigned long chanNum);

Function: Save all user-configurable variables' current values to the **initFile** (e.g. MPEGIO2.ini) under the folder **MPEGIO2** SDK resides(if the **initFile** exists)
Input Parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Return: True for success.
Note: This function must be called after **MPEGIO2_initSDK** has been called.

MPEGIO2_API bool **MPEGIO2_createUnicodeInitFile**(void);

Function: Delete the **iniFile**, recreate it as Unicode file then write "totalChans=totalChans' value" under section [System]
Return true for success
Note 1: This function can be called without first calling **MPEGIO2_initSDK**()
Note 2: Call this function before **MPEGIO2_endSDK** if the application has Unicode strings such as Unicode Overlay Text.

4.16 PCB IC Reset Functions

These functions will cause hardware reset on some crucial IC chips on the **MPEGIO2** card, great caution is needed when using them: calling these functions will cause the chips to revert back to their hardware default status with all SDK loaded register values lost.

```
MPEGIO2_API unsigned long MPEGIO2_loadDefaultRegs(    unsigned long chanNum,
                                                      unsigned long TVFormat,
                                                      unsigned long ICNum,
                                                      bool restoreBoxItems = false,
                                                      char *regFile = 0);
```

Function: Load an IC's Default Register Values on a channel using the default register file or a caller supplied register file
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **TVFormat**: 1 for NTSC, 2 for PAL, only used when **regFile** is invalid
Input parameter **ICNum**: 0=OSD and Video Input IC,
1=the 5Th Video Input Source (VIN4) IC,
2=Video Output IC.
Input parameter **restoreBoxItems**: Only applicable for OSD IC(**ICNum** = 0): if true, after loading values, will restore all OT_BOX type overlay items to their previous status prior to the register values loading. Default is false(not restore box overlay items' status: some of them, such as colour, will be lost).
Input parameter **regFile**: If is a valid file name inc full path and containing the register number-value pairs for ICNum chip, will use this file instead of the default register value file to load into **ICNum** chip's registers.
Return: The number of Register Number + Value pairs loaded into ICNum IC. Return 0 for failure or loading nothing.
Note: After successfully loading the OSD IC's default register values all Overlay settings on video are erased.
The 5Th Input IC and Video Output IC will also lose their current settings.

```
MPEGIO2_API bool  MPEGIO2_resetADevice(unsigned long chanNum,
                                       unsigned long devNum,
                                       unsigned long delay = 20);
```

Function: Reset one device on the channel then clearing the reset condition after some delays
Input parameter **chanNum**: 0 ~ totalChans-1: the **MPEGIO2** channel number
Input parameter **devNum**: 5=Reset A/V Input and OSD IC, 7=Reset MPEG Codec IC
Input parameter **delay**: millieseconds between reset and clearing the IC
Return true for success.

```
MPEGIO2_API bool  MPEGIO2_resetAVPreviewIC(unsigned long chanNum,
                                           unsigned long delay = 50000);
```

Function: Reset the Video/Audio Preview IC's Register Values to their default ones (as hard-wired in Device Driver Codes)
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **delay**: the delay looping times after writing each register value to the Video/Audio Preview IC during chip initialization time, default is 50000.
This delay is to fix audio preview and audio A/D conversion output (via I2S) unstability problem, usually there is no need to modify it unless there is a unstability problem in previewing/encoding audio , Higher value tends to be more stable although slow down this function call.
(the unstability problem also affects I2S audio output from Preview IC to MPEG Encoding IC)
Return: true for success
Note: This function reset preview video's colour, margins etc. settings to device driver default values.

```
MPEGIO2_API bool MPEGIO2_getRegWriteDelay(unsigned long chanNum, unsigned long &delay);
```

Function: Get Video/Audio Preview IC's Register Write Delay Value on an MPEGIO2 channel
Input parameter chanNum: 0 ~ totalChans-1: the channel number
Output parameter **delay**: the delay looping times after writing each register value to the Video/Audio Preview IC during chip initialization time, as set by **MPEGIO2_resetAVPreviewIC**
Return: true for success.

```
MPEGIO2_API bool  MPEGIO2_powerdown5ThInput(unsigned long chanNum, bool powerdown);
```

Function: Power-down or power-up the 5^{Th} Video Input Device Decoder IC:
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **powerdown**: true to power-down the 5^{Th} Video Input Device Decoder IC, false(default) to power it up.
Note: The 5^{Th} Input controls the **VIN4 RCA** and **VIN5 S-Video** Input Sockets on the **Breakout Box**.

MPEGIO2_API bool **MPEGIO2_powerdown5ThInputStatus**(
unsigned long chanNum, bool &powerdown);

Function: Return power-down or power-up status for the 5^{Th} Video Input Device Decoder IC.
Input parameter **chanNum**: 0 ~ totalChans-1: the MPEGIO2 card number
Output parameter **powerdown**: true the 5^{Th} Video Input Device Decoder IC is Powered Down, false is Powered Up.
Return True for Success.

MPEGIO2_API bool **MPEGIO2_resetMPEGCodecIC**(unsigned long chanNum,
unsigned long delay = 0);

Function: Reset The MPEG Encoding IC for a Channel
Input parameter **chanNum**: 0 ~ totalChans-1: the channel number
Input parameter **delay**: milliseconds to delay between resetting and clearing the IC
Return: true for success
Note: This function is the same as calling **MPEGIO2_resetADevice** with parameter devNum ==7.

MPEGIO2_API bool **MPEGIO2_recreateMPEGCodecIC**(unsigned long chanNum,
bool verifyFW = true);

Function: Thouroughly recreate the **MPEGIO2** channel's MPEG Encoding IC's entire software structure including re-download its firmware and reset its pins.
Input Parameter **chanNum**: MPEGIO2 Channel Number, must be between 0~totalChans-1
Input parameter **verifyFW**: If false means not verifying the downloaded firmware to the MPEG Codec IC: this will lead to slightly faster initialization of the Codec IC although risking not finding a possibly faulty IC or faulty associated SDRAM on PCB. Default is true: always verifying downloaded firmware.
Return true for success.
Note 1: If the channel is encoding/decoding all of these will be stopped before recreating the IC's software structure.
Note 2: The channel's MPEG Codec IC must be available when calling this function otherwise false is returned.
Note 3: This function is more thorough than calling fucntion **MPEGIO2_resetMPEGCodecIC** in terms of re-initializing and resetting the MPEG Codec IC on an **MPEGIO2** channel.

MPEGIO2_API bool **MPEGIO2_ResetPCIeBus**(unsigned long chanNum);

Function: Reset the PCIe Bus on which this **MPEGIO2** channel(chanNum) resides.
Input Parameter **chanNum**: **MPEGIO2** Channel Number, must be between 0~totalChans-1
Return true on success.
Note: This reset affects both the left and right **MPEGIO2** channels on the same PCIe bus (since they are on the same card).

4.17 Callback Setup Functions

MPEGIO2 SDK allows callback functions to be defined by application software and their addresses passed to the SDK(this is called "Setup a Callback Function"): these callback functions can then be called by the SDK at various times such as MPEG encoding/decoding, Digital Input Status Change, mouse button down in Video Preview Window, etc., giving the application software capability to process the MPEG data and respond to status changes in real-time.

The SDK's C header file **MPEGIO2.h** defines several callback function types:

```
// Callback function types used during MPEG Encoding or Decoding process:
typedef int (_cdecl    * MPEGIO2_callback_streamC)(unsigned long chanNum, unsigned char *buf, unsigned long length);
// used by C/C++ etc languages
typedef int (__stdcall * MPEGIO2_callback_streamS)(unsigned long chanNum, unsigned char *buf, unsigned long length);
// used by C#/VB etc languages
```

MPEGIO2_callback_streamC or **MPEGIO2_callback_streamS** type of callback functions are setup by using SDK functions **MPEGIO2_setEncodeCBC/MPEGIO2_setEncodeCBS** or **MPEGIO2_setDecodeCBC/MPEGIO2_setDecodeCBS** as parameter **fC** or **fS** so that these application-defined functions can be called back by **MPEGIO2** SDK during a channel's MPEG encoding or decoding process when a chunk of encoded or decoded MPEG data is available:
each time when being called, these application-defined **MPEGIO2_callback_streamC** or **MPEGIO2_callback_streamS** type functions can process this chunk of encoded or decoded data.

```
// call back function types used when a Video Input or Digital Input interrupt occurred:
typedef void (_cdecl  * MPEGIO2_callback_INTC)
                    (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
typedef void (__stdcall * MPEGIO2_callback_INTS)
                    (unsigned long chanNum, unsigned long type, unsigned long num, unsigned char *MDPtr);
```

Note: See **"MPEGIO2_setVideoInputNotify"** and "**MPEGIO2_setDIONotify"** described in **"Digital I/O Functions"** Section for set up or clear of MPEGIO2_callback_INTC/ MPEGIO2_callback_INTS type callback functions**.**

```
// call back function types used when a mouse button is down in video preview window
typedef void (_cdecl    * MPEGIO2_callback_mouseDownC)(unsigned long chanNum, unsigned short wparam, unsigned long lparam);
// used by C/C++ etc languages
typedef void (__stdcall * MPEGIO2_callback_mouseDownS)(unsigned long chanNum, unsigned short wparam, unsigned long lparam);
// used by C#/VB etc languages
```

Important common points about the callback functions and their setup/clear in this section:

Note 1: If a callback setup function (**MPEGIO2_setEncodeCBC**, **MPEGIO2_setDecodeCBS**, etc.) is called twice with different callback function address values, only the last call's callback function address will be called by the SDK.

Note 2: The SDK will not check if the callback function address passed to callback setup function is correct, so passing illegal memory address can cause system crash.

Note 3: Passing 0 as callback function address to SDK's callback setup functions will clear the callback.

Note 4: By default there is no callback function during any SDK operations.

Note 5: See **MPEGIO2.h** file for detailed description of all callback function types inc. their parameters.

Note 6: C++ for MPEGIO2.exe, C#.net source codes included with the SDK have callback functions setup/clear examples.

Note 7: Callback functions must not do time-consuming processing in their main body: they should return to caller asap.

MPEGIO2_API bool **MPEGIO2_setEncodeCBC**(unsigned long chanNum,
MPEGIO2_callback_streamC fC);

Function: Set up "**int (_cdecl * MPEGIO2_callback_streamC)**" type callback function for encoding process on a channel

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

Input parameter **fC**: If not NULL, must point to an **MPEGIO2_callback_streamC** type function that will be called back by **MPEGIO2** SDK during MPEG encoding
If this parameter is NULL (0), the callback function for encoding will be cleared.
(it is not an error if 0 is passed and there is no previous call of this function with non-zero **fC**)

Return true for success.

MPEGIO2_API MPEGIO2_callback_streamC **MPEGIO2_getEncodeCBC**(unsigned long chanNum);

Function: Return "**int (_cdecl * MPEGIO2_callback_streamC)**" type callback function address for encoding process on a channel if that has been set up previously by function **MPEGIO2_setEncodeCBC**, if not, return 0.

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1.

MPEGIO2_API bool **MPEGIO2_setEncodeCBS**(unsigned long chanNum,
MPEGIO2_callback_streamS fS);

Function: Set up "**int (__stdcall * MPEGIO2_callback_streamS)**" type callback function for encoding process on a channel.

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

Input parameter **fS**: If not NULL, must point to an **MPEGIO2_callback_streamS** type function that will be called back by **MPEGIO2** SDK during MPEG encoding

If this parameter is NULL (0), the callback function for encoding will be cleared.

(it is not an error if 0 is passed and there is no previous call of this function with non-zero **fS**)

Return true for success.

MPEGIO2_API MPEGIO2_callback_streamS **MPEGIO2_getEncodeCBS**(unsigned long chanNum);

Function: Return "**int (__stdcall * MPEGIO2_callback_streamS)**" type callback function address for encoding process on an **MPEGIO2** Channel if that has been set up previously by **MPEGIO2_setEncodeCBS**, if not, return zero.

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

MPEGIO2_API bool **MPEGIO2_setDecodeCBC**(unsigned long chanNum,
MPEGIO2_callback_streamC fC);

Function: Set up or clear a "**int (_cdecl * MPEGIO2_callback_streamC)**" type callback function for decoding process on an **MPEGIO2** Channel

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

Input parameter **fC**: If not NULL, must point to an **MPEGIO2_callback_streamC** type function that will be called back by **MPEGIO2** SDK during MPEG decoding.

If this parameter is NULL (0), the callback function for decoding will be cleared.

(it is not an error if 0 is passed and there is no previous call of this function with non-zero **fC**)

Return true for success.

MPEGIO2_API MPEGIO2_callback_streamC **MPEGIO2_getDecodeCBC**(unsigned long chanNum);

Function: Return "**int (_cdecl * MPEGIO2_callback_streamC)**" type callback function address for decoding process on a channel if that has been set up previously by **MPEGIO2_setDecodeCBC**, if not, return zero.

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1.

MPEGIO2_API bool **MPEGIO2_setDecodeCBS**(unsigned long chanNum,
MPEGIO2_callback_streamS fS);

Function: Set up or clear a "**int (__stdcall * MPEGIO2_callback_streamS)**" type callback function for decoding process on an **MPEGIO2** Channel

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

Input parameter **fS**: If not NULL, must point to an **MPEGIO2_callback_streamS** type function that will be called back by **MPEGIO2** SDK during MPEG decoding

If this parameter is NULL (0), the callback function for decoding will be cleared.

(it is not an error if 0 is passed and there is no previous call of this function with non-zero **fS**)

Return true for success.

MPEGIO2_API MPEGIO2_callback_streamS **MPEGIO2_getDecodeCBS**(unsigned long chanNum);

Function: Return "**int (__stdcall * MPEGIO2_callback_streamS)**" type callback function address for decoding process on a channel if that has been set up previously by **MPEGIO2_setDecodeCBS**, if not, return zero.

Input parameter **chanNum**: **MPEGIO2** channel number, between 0 ~ totalChans-1

MPEGIO2_API void **MPEGIO2_setMouseDownCBC**(MPEGIO2_callback_mouseDownC fC);

typedef void (_cdecl * MPEGIO2_callback_mouseDownC)(unsigned long chanNum, unsigned short wparam, unsigned long lparam);

MPEGIO2_API void **MPEGIO2_setMouseDownCBS**(MPEGIO2_callback_mouseDownS fS);

typedef void (__stdcall * MPEGIO2_callback_mouseDownS)(unsigned long chanNum, unsigned short wparam, unsigned long lparam);

Function: **MPEGIO2_setMouseDownCBC** or **MPEGIO2_setMouseDownCBS** setup or clear callback function of type **MPEGIO2_callback_mouseDownC** or **MPEGIO2_callback_mouseDownS** that will be called when a mouse button is pressed inside video preview window:

If **fC** or **fS** is a valid callback function then they will be called by **MPEGIO2** SDK when a mouse is pressed inside the video preview window. If **fC** or **fS** is NULL then the callback function is cleared (default).When **fC** or **fS** is called back, the **chanNum** indicates the **MPEGIO2** channel whose video preview area covers the mouse cursor.

The **wparam** and **lparam** parameters in functions pointed to by **fC** or **fS** will be the same as in WM_LBUTTONDOWN/WM_RBUTTONDOWN/WM_MBUTTONDOWN Windows SDK Messages plus the **MPEGIO2** channel number
in high bits[15:12] of **wparam** which indicates in which **MPEGIO2** channel's area the mouse button was clicked.
Note: These functions are only applicable when there is only one Video Preview Window for the entire **MPEGIO2** SDK. When on Windows 7 or above and each channel has created its own Video Preview Window, do not use these functions(then each Video Preview Window can find its own mouse clicking position through e.g. MS Windows SDK functions using the Video Preview Window Handle received from calling **MPEGIO2_initSDK** function).

```
MPEGIO2_API MPEGIO2_callback_mouseDownC     MPEGIO2_getMouseDownCBC(void);
MPEGIO2_API MPEGIO2_callback_mouseDownS     MPEGIO2_getMouseDownCBS(void);
```

Function: Return the callback function address set up previously by calling **MPEGIO2_setMouseDownCBC** or **MPEGIO2_setMouseDownCBS.** Return 0 if no callback was set up.

4.18 PCB IC Register Access Functions

MPEGIO2 SDK can directly write and read all major on-board ICs' registers through simple function calls: detailed syntax and IC register maps can be obtained from support@inventa.com.au.

5. SDK to Application Messages

The **MPEGIO2 SDK** can pass SDK-defined messages to the Window handle the application software passed to function **MPEGIO2_initSDK** as parameter "**parentWnd**". These messages are defined in header file "**MPEGIO2.h**", in the form of "MPEGIO2_MSG_ XXX", such as:

```
#define MPEGIO2_MSG_SDK_EXIT              WM_USER + 21 // The SDK has exited
#define MPEGIO2_MSG_ERR_START_ENCODE      WM_USER + 59 // failed to start MPEG IC to encode, WPARAM Contains MPEGIO2 channel no.
```

Many such messages are very useful in async. operations, e.g. **MPEGIO2_MSG_DECODE_END** tells the Application an MPEG file playback ended, **MPEGIO2_MSG_TIMER_COUNTER_ENDED** tells the Application an Overlay counter ends its counting, etc.
Note WM_USER usually is defined by the MS Windows SDK as 0x0400 (in Winuser.h)

6. SDK Functions Calling Sequence

Most **MPEGIO2 SDK** functions must be called between calling the SDK start function **MPEGIO2_initSDK** and calling the SDK end function **MPEGIO2_endSDK**. However, all the **"initFile"** Functions (see the **"initFile" Functions** Section), and some Generic SDK Functions can be called without calling **MPEGIO2_initSDK** first, such as:

MPEGIO2_getSDKVer,
MPEGIO2_getPCBVer,
MPEGIO2_setSoftwareName,
MPEGIO2_getSDKPath.

The **MPEGIO2 SDK** requires function **MPEGIO2_initSDK** to return a positive integer to operate most of its functions, which means the PC has at least one **MPEGIO2** channel properly connected and device drivers installed.

7. SDK Installation & Running Environment

7.1 Install the SDK

To run any executable program on a Windows PC that calls functions in the **MPEGIO2 SDK**:

(1) Copy these **.dll** files in the **MPEGIO2** Setup CD's **SDK\lib\WinXP(Win7)\Debug** or **SDK\lib\WinXP(Win7)\Release** folder to a library searchable path such as C:\Windows\Systen32, or to the same folder as the executable program:

- **Dynamic Linking Libraries** (Note use different **MPEGIO2.Dll** for WinXP and Win7):

```
SDK\lib\WinXP\Debug\MPEGIO2.dll or  SDK\lib\WinXP\Release\MPEGIO2.dll   // Main DLL for WinXP
SDK\lib\Win7\Debug\MPEGIO2.dll   or  SDK\lib\Win7\Release\MPEGIO2.dll    // Main DLL for Win7
vwSDKD.dll (Debug version) or vwSDKR.dll(Release Version),               // Codec IC SDK DLL
MulticastSender.dll,                                                     // UDP MultiCast DLL
UdpSender.dll,                                                           // UDP UniCast DLL
CCommPort.dll.                                                           // Serial Comm. DLL
```

(2) The following data files must be in the same folder as the executable program:

- **IC Register Init Files:**
  DefaultOpt.ini
  DefaultOpt_ts.ini
  OSDValsPAL.ini
  OSDValsNTSC.ini
  Input5ValsPAL.ini
  Input5ValsNTSC.ini
  OutputPAL.ini
  OutputNTSC.ini
- **IC Firmware Files**:
  ac3ps_codec.sre
  boot.sre
  g7xxps_codec.sre
  mp3eps.sre
  pscodec.sre
  tscodec.sre
- **Other Data Files:**
  MPEGIO2.bmp – Needed for Loading Overlay Colour Lookup Table
  MPEGIO2.ico

(3) The Device Driver files must be in the same folder as the utility programs **DevCon.exe** (from Microsoft Windows DDK) **SetupDrv.exe** and **ResetDrv.exe:**

- **Device Driver Files:**
  Drv2010.sys
  Drv7134.sys
  Drv2010.inf
  Drv7134.inf

Note: Device Driver Files for Windows XP, Windows 7 and Windows 8 are different under the WinXP/Win7/Win8 folders on the Setup CD: these must be installed onto the target PC appropriately according to the actual Windows' version. During the **MPEGIO2.exe**'s installation, the appropriate Device Driver files for the correct Windows are installed onto the **MPEGIO2** program group: any application program using **MPEGIO2** SDK should do the same by copying the correct driver files to the target PC together with **DevCon.exe, SetupDrv.exe** and **ResetDrv.exe.**

7.2 Create Applications with the SDK

- Using C or C++ Languages outside Windows **.Net** Framework Programming Environment:
  After copying the above listed files**,** include the header file **MPEGIO2.h** and link the library **MPEGIO2.lib** with your source code will create an executable program that can dynamically call functions in **MPEGIO2.dll**.

- Using Microsoft Windows **.Net** Framework Programming Languages:
  MPEGIO2 SDK comes with a Windows' **.Net** Class Library **MPEGIO2API.dll**, that wraps most of the **MPEGIO2 SDK** library functions declared in **MPEGIO2.h**: sample applications calling these functions from C++ .Net, C#.Net and VB.Net languages are also included with source codes, together with the C# source code of the class library **MPEGIO2API.dll** which users can modify to include more functions from **MPEGIO2.h.**

- Other Programming languages that cannot directly use C-styled function declarations inside **MPEGIO2.h** need individually declare any **MPEGIO2.dll** functions to be called, using the function prototypes in **MPEGIO2.h,** as demonstrated in the (non .Net) VisualBasic sample project "**MPEGIO2VBApplication**" and source code.

- **Compiler Switch** for Windows XP and Windows 7(or above): an "**ABOVE_WINXP**" compiler switch must be used differently when compiling under MS Windows XP and Windows 7(or above): under Windows XP, **ABOVE_WINXP** must be defined as = 0 (or not defined at all), while under Windows 7 or above, **ABOVE_WINXP** must be defined as non-zero – this has been done in the C++ project for the **MPEGIO2.exe**, as well as in the sample VB, C++.Net, C#.Net, VB.Net(under Compile->Advanced Compile Options->Custom constants) projects and the **MPEGIOAPI.dll** C# .Net Class Library project(under Build->Conditional compilation symbols). Note in the .Net VisualStudio projects **ABOVE_WINXP** must be defined under Windows 7 or above, while under Windows XP it must not be defined --- this is different from the VS projects used for C/C++ outside the .Net environment: e.g. in the **MPEGIO2.exe** project, where under Windows 7 or above **ABOVE_WINXP** is defined as == 1, while under Windows XP **ABOVE_WINXP** is defined as == 0. The direct consequence of using this switch is: the **MPEGIO2.dll** and the **MPEGIOAPI.dll** created and used under Windows XP are different from that created and used under Windows 7 or above, and also the executables (**MPEGIO2.exe** etc.) created will be different between under WinXP and under Win7 or above.

- To run programs created with **MPEGIO2** SDK, all the "**Dynamic Linking Libraries**" **.dll** files listed above must be in a library search path, preferably in the same folder as the executable program itself: this folder must also have the previously listed **"IC Register Init Files", "IC Firmware Files"** and the **"Other Data Files".**

- To run the sample C++/VB/C# programs (inc. the **MPEGIO2.exe**) created with **MPEGIO2** SDK, the target PC must install the MS Visual C++ 2008 Redistributable Package for x86 CPU "**vcredist_x86.exe**" from http://www.microsoft.com/download/en/details.aspx?id=11895 (**MPEGIO2.exe** does this in its Start.bat file).

- Running Debug version application software created with debug version **MPEGIO2** SDK (**MPEGIO2.Dll** and **vwSDKD.Dll** under folders SDK\lib\WinXP\Debug and SDK\lib\Win7\Debug) need the following extra files (from the SDK\lib\WinXP\Debug or SDK\lib\Win7\Debug folder) to be in the same folder as the application program, to avoid the Windows' "The application failed to initialize properly (0xc0150002) " etc. side-by-side configuration problem(Note if VisualStudio 2008 is installed then most or all of these files won't be needed):

  Microsoft.VC90.DebugCRT.manifest,
  Microsoft.VC90.DebugMFC.manifest,
  mfc90D.dll,
  msvcm90D.dll,
  msvcp90D.dll,
  msvcr90D.dll,
  D3dx9D_42.dll(Win7 or above),
  MSVCRTD.DLL(WinXP only).

8. Sample Source Codes

8.1 Using VisualStudio 2008(VC++9.0):

Under the folder **SDK\src** on the **MPEGIO2** Setup CD there are Microsoft Visual C++, VB, C++.Net, C#.Net, VB.Net sample source codes and their Visual Studio (VS Pro 2008) projects that can be used as reference to build application programs utilizing the **SDK**.

The C++ source codes under **SDK\src\MPEGIO2XP** and **SDK\src\MPEGIO2W7** folders implement the entire application program **MPEGIO2.exe** shipped with the **MPEGIO2** hardware for WinXP and Win7 respectively, and compiled and linked with MS VisualStudio 2008 Pro(VC++ 9.0). The source files under these two folders are identical, but their VS 2008 projects used different settings for compiler preprocessor **ABOVE_WINXP**: 0 for Windows XP, 1 for Windows 7 or above.

The C#.Net source code under **SDK\src\DotNet\ MPEGIO2ClassLibrary** folder implements the Windows' .Net Class Library **MPEGIO2API.Dll** used by all .Net sample programs.

The sample C#.net project and source code implement callback functions written in C# and passed to the **MPEGIO2** SDK to be called back during MPEG encoding and mouse button clicking, it also demonstrates the **Downloadable Font** functions in the SDK which can repeatedly display text instantly by first downloading the entire alphabet of a text font.

The sample C++.net project and source code allow options to show multiple separately floating Video Preview Windows for multiple **MPEGIO2** channels under Windows 7 or above.

A sample Serial Comm. Port Library source code is under **SDK\src\CCommPort** folder which creates the previously mentioned **CCommPort.dll** library that can be used to build Comm. Port applications.

To compile and link the sample source codes, do the following differently under WinXP and Win7:
(1) use different Compiler Switch "**ABOVE_WINXP**":
C++, VB must define **ABOVE_WINXP**=0 for WinXP, and define **ABOVE_WINXP**=1 for Win7;
C++.Net, C#.Net(inc. C# for MPEGIO2ClassLibrary) must Not define **ABOVE_WINXP** for WinXP, and must define **ABOVE_WINXP** for Win7.

(2) use different Output path to hold the created executables:
WinXP uses SDK\lib\WinXP\Debug or SDK\lib\WinXP\Release,
Win7 uses SDK\lib\Win7\Debug or SDK\lib\Win7\Release.

8.2 Using VisualStudio 2002(VC++7.0):

The C++ source codes under "**SDK\src\MPEGIO2 for VS 2002**" folder implements the entire application program **MPEGIO2.exe** shipped with the **MPEGIO2** hardware compiled and linked with Microsoft VisualStudio 2002 (VC++ 7.0): the source files under this folder are identical with those under the VisualStudio 2008 folders (**SDK\src\MPEGIO2XP** and **SDK\src\MPEGIO2W7**) except several conditional compilation lines enclosed within the #ifdef **USE_VS_2002** …#endif to handle syntax differences between VC++7 and VC++9, and the conditional compilation preprocessor "**USE_VS_2002**" is defined, also library **Ws2_32.lib** need to be explicitly linked in. Note this project has **ABOVE_WINXP** defined as 0 for Windows XP. To compile and link under Windows 7 or above **ABOVE_WINXP** must be defined as 1. Unlike the VS 2008 projects, this project output to its own Debug and Release folders and the created **MPEGIO2.exe** program (with identical functions as those created using VS 2008) can be run there directly.

8.3 Using VisualStudio 6.0(VC++6.0):

The C++ source codes under "**SDK\src\MPEGIO2VC6WinXP**" folder implements a simple program MPEGIO2Test.exe compiled with Microsoft VC++ 6.0 under VisualStudio 6 and linked with **MPEGIO2.lib**: MPEGIO2Test.exe can test live video preview with **MPEGIO2** SDK on Windows XP.

9. SDK Release Notes

In current SDK release the following limitations are imposed:

(1) MPEG Audio encoding has no Audio 2 support;

(2) MPEG Decoding does not support Network Stream as Input;

(3) Microphone Audio Input is not available.

10. MPEGIO2 Hardware Specification

Host Interface: 1X PCI-Express Card
Maximum Power Consumption: < 10 Watts
Video Input: 6 X Composite (RCA), 2 X S-Video (4-Pin Mini-DIN)
Video Output (for Real-time Monitoring and MPEG Decoding): 4 X Composite (RCA), 2 X S-Video
Audio Input: 4 X Line-in 3.5mm Stereo Mini Socket
Audio Output: 2 X Line-out 3.5mm Stereo Mini Socket
Encoded/Decoded Video Formats: MPEG-1, MPEG-2 MP@ML, MPEG-4 Simple Profile@L1,L2 & L3, with extensions to full-D1,H.263. I, IP, IBP, IBBP Frames. PES & ES. Support Both Program Stream and Transport Stream Encoding & Decoding
Constant Bit Rate (CBR) and Variable Bit Rate (VBR) Encoding
Video 4:2:2 to 4:2:0 Conversion
Video Inverse telecine (3:2 pulldown)
Video Encoding Frame Rates: 23.976fps, 24 fps, 25fps, 29.97fps, 30fps, 50fps, 59.94fps, 60fps
Video Encoding Bit Rates: 128 Kbps ~ 15.00 Mbps
Video Encoding / Decoding Resolution in Pixels: Horizontal 176, 352, 480, 528, 544, 640, 704, 720, Vertical 120, 144, 240, 288, 480, 512, 608, 576
Video Encoding Aspect Ratio: 4:3, 16:9, 2.21:1, Square Pels
Audio Encoding Format: MPEG1 Layer 1, Layer 2, Layer 3(MP3), MPEG-2 audio, AC3, AAC, G.711, G.723, G.726, G.729
Audio Sampling Rates: 32KHz, 44.1KHz, 48Khz
Audio Encoding Bit Rates: 0 to 448Kbps, inc. 64Kpbs, 128Kpbs, 160Kbps, 192Kbps, 224Kbps, 256Kbps, 320Kbps, 384Kbps. etc.
Digital Input/Output Pins: 8 Pins (4 for each Channel): each Pin responds to -0.5V ~ +0.3V as Low Status, +0.7V ~ +5.5V as High Status, with maximum current 25mA per Pin, and a total current sourced by all Pins limited to 160mA. 1 Ground Pin Common for All I/O Pins.
PCIe Card Dimension: Length 230mm, Height 145mm