MPEGIO SDK User Manual

For
Writing Multiple-Stream Hardware MPEG1/2/4 Video Encoding / Decoding Software

Version 1.0.8.6

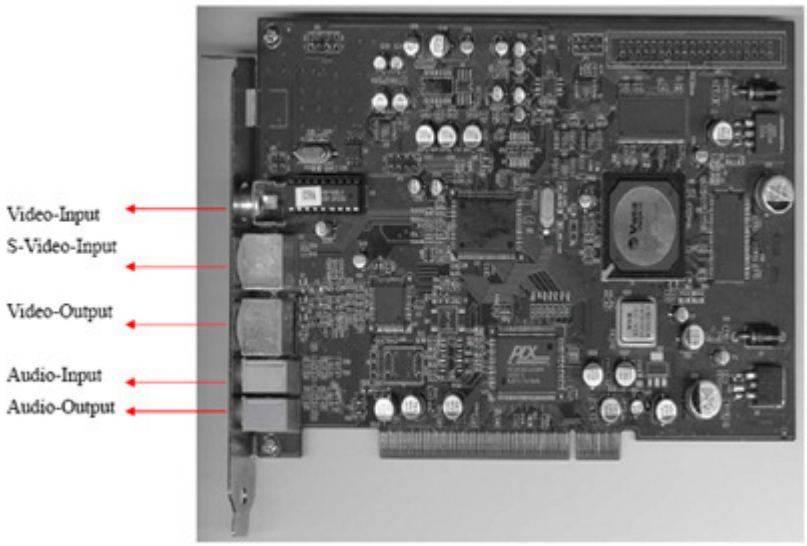


Table of Contents

1. Introduction:

MPEGIO SDK (Software Development Kit) is for rapidly developing application software using **MPEGIO** hardware MPEG1/2/4 encoder/decoder card. **MPEGIO SDK** shields the software developers from coding complicated hardware interfacing and DirectShow/DirectX functions, by supplying highly integrated and extremely easy to use C-styled function calls. Using **MPEGIO SDK**, even entry-level software developers or students can quickly write application software to control one or more MPEGIO cards on the same PC to capture, play, transcode and stream high-quality MPEG video for various applications, using C++, VB, Delphi, Java or any other programming languages that can call external DLL library functions. Programmers familiar with Microsoft DirectShow SDK can also use their own DirectShow filter connected graphs to preview **MPEGIO** video while simultaneously calling **MPEGIO SDK** functions to encode, record and stream MPEG video.

For details on the **MPEGIO PCI card**, please refer to the "**MPEGIO User Manual**", here is a summary of the PCI card's main features:

- **Realtime capture superior-quality video & audio as MPEG1, MPEG2 or MPEG4 video files**
- **Playback superior-quality video to TV/VCR through on-board video & audio output sockets**
- **Realtime transcode (convert) MPEG files to different parameters(bitrate, frame sizes, etc)**
- **Realtime Stream hardware encoded/decoded MPEG video to any TCP/IP network address**
- **Video Streaming Can Be Started in Video Encoding, Decoding or Transcoding Mode**
- **Wide-Range Capture Data Rates 50Kbps~15Mbps for MPEG1, MPEG2 & MPEG4 formats**
- **1~10 Cards can simultaneously run on the same PC for concurrent encode/decode/transcode**
- **Allow Multiple Aspect Ratio Encoding: 4:3, 16:9(Wide-Screen), and Square PEL**
- **Forever Video & Audio Sync during video preview, capture, playback and in the video files**
- **Realtime split captured video into multiple files at different size/ time,**
- **Instantly Start Video Capture at User's Request (Button press) without lengthy delays**
- **Start Video Capture as Video Signal Appears**
- **Stops Video Capture after Video Signal Disappears for a period of user-defined time**
- **Allow Pause and Resume during encoding without closing the current recording MPEG file**
- **Arbitrary User Data can be realtime inserted into the currently encoded/streamed MPEG video**
- **Capture still images**
- **Capture DVD or arbitrary MPEG2, MPEG4 or MPEG1 video files**
- **Capture and Stream MPEG2 and MPEG4 video in either Program Stream or Transport Stream Format**
- **Captured DVD files can be used to burn DVD movie disk without re-compression**
- **Realtime overlay multiple text lines and graphics on MPEG videos being decoded or transcoded**
- **Realtime output text/graphics overlaid video directly to external TV**
- **Realtime preview incoming video and audio with video in sync with incoming signal**
- **Realtime preview incoming video and audio with video & audio perfectly synchronized**
- **PAL and NTSC video format supported**
- **SVideo and Composite Video Input and Output**
- **Stereo Audio Input and Output**
- **Hardware-encoding chipset built-in, low system resources required**
- **Support live colour change on incoming video**

2. MPEGIO.DLL & MPEGIO SDK

The **MPEGIO SDK** is based on a dynamic linking library **MPEGIO.DLL,** which supports all the **MPEGIO SDK** function calls the application software might use. **MPEGIO.DLL** will need to be installed on the target PC running your application software.

The **MPEGIO.DLL** system has several main operation modes, including **Preview**, **Encode**(**Record**), **Decode**(**Playback**), **Transmit**, **Transcode**, etc. When first time started, the DLL is normally in "**Preview**" mode, which simply displays the incoming video, if any, on the **Video Window**. From thereafter, the application software is free to set the operation into other modes.
The **Video Window** is where **MPEGIO.DLL** displays the video contents all the time. The Video Window can be setup by your application software as any window's child window, to be freely manipulated by your application software, such as maximizing, hiding, text overlay, etc.

3. MPEGIO SDK Function List:

-- char * **MPEGIO_getSDKVer**(void);
Function: Return the version of the MPEGIO SDK as a string.

-- void **MPEGIO_setSoftwareName**(char *name);
Function: Set the application software name to user-defined string "name", so that all Dialog boxes(inc. the "Setup Dialog" box as described in **MPEGIO_startSetupDialog** function below) etc will use the new name in their title. If not set, the application name is referred to as "MPEGIO"
Parameter: "name" must point to a null-terminated string for the new software name.

-- int **MPEGIO_getTotalMPEGIOCards**(void);
Function: Return the total number of installed MPEGIO cards in the PC, 0 means no card.
Note: This function depends on the properly installed device drivers of the MPEGIO cards, not the physical cards themselves. So if one MPEGIO card's drivers are not installed properly then that card will not be counted as being present in the PC, even though that MPEGIO card sits in the PCI slot.

-- bool **MPEGIO_initAllCards**(char *testfile, HWND parentWnd, bool forceInit);
Function: Initialize all installed MPEGIO cards in the PC. Return non-zero for success.
Parameters:
testfile: File name of a valid MPEG file previously encoded by MPEGIO card. It is recommended to use appropriate PAL or NTSC format MPEG file according to your primary TV system in use. This file will be used by the MPEGIO SDK to test the installed MPEGIO cards. The C++ sample folder include testpal.mpg and testntsc.mpg sample files that you can use.
parentWnd: any valid window's handle served temporarily as a parent window during the cards testing process. This cannot be NULL.
forceInit: If true, the initialization will always happen. If false, the initialization will only happen if no initialization has been performed previously. After new card or driver is added/removed, you should Call **MPEGIO_initAllCards**() with forceInit = TRUE at least once.
Note: **1.** This function must be called at least once with forceInit = TRUE, after any new MPEGIO cards and/or their drivers are installed or de-installed on a PC. Only after a successful call to this function can other **MPEGIO SDK** functions be called, in particular the call to **MPEGIO_start** function must be preceded by a successful call to this function.
2. Depending on the number of MPEGIO cards in your PC, this function can take some time to finish, during this time your application software will not gain any control of the system and you/users are advised not to operate mouse and keyboard.

-- HWND **MPEGIO_start**(int BoardNumber = 0,
HWND ParentWindow = NULL,
bool notReadInitFileAtStart = false,
int listAllChildWnd = 0,
char *parentWndTitle = NULL,
char *parentWndClass = NULL,
bool parentHasParent = false,
int parentWndX1 = -1, int parentWndY1 = -1,
int parentWndX2 = -1, int parentWndY2 = -1,
bool parentWndHide = false);

Function: Start one **MPEGIO** Card.
Parameters:

BoardNumber: Which MPEGIO card to start. If there is only one card in the PC, pass 0. When there are N > 1 cards installed, pass value between [0, N-1].

ParentWindow: any valid window's handle served as a parent window for this started MPEGIO card's video window (live video will appear inside **ParentWindow** if **parentWndHide** is false and **ParentWindow** is visible).

If this parameter is **a valid window handle**, then this window will be used as the parent window for the video window started by this MPEGIO card, and the following parameters' values will be ignored:

```
char *parentWndTitle,
char *parentWndClass,
bool parentHasParent,
int parentWndX1, int parentWndY1,
int parentWndX2, int parentWndY2.
```

If this parameter is **not a valid window handle**, e.g., if it is NULL (when application software cannot find a suitable window handle to pass over a NULL is handy), then:

(1) If **listAllChildWnd** is **1 or 2**, the dialog "**Select Parent Window for Video Window**" will appear, see comments below next to the **ListAllChildWnd's** explanation. In this situation the parameters

```
parentWndTitle,
parentWndClass,
parentHasParent,
parentWndX1, parentWndY1,
parentWndX2, parentWndY2,
```

will also be ignored (same as when **ParentWindow** is a valid window handle).

(2) If **listAllChildWnd** is **3, no dialog will appear, MPEGIO video window will have no parent window:** it will be created as top-level window that can be resized, moved independently. The video window's initial position will be remembered each time when the MPEGIO card if stopped and this position will be reused when the card is re-started next time. When **listAllChildWnd** is **3**, if the parentWndHide = TRUE and the **MPEGIODX.EXE** file (from SDK's setup CD) is in C:\Windows\System32, then the MPEGIO video window cannot be manually closed or killed, it will be killed automatically when **MPEGIO_stop()** is called on that MPEGIO card.

(3) If **listAllChildWnd** is **0**, no dialog will appear, and a search among all existing windows will start to find a matching window that satisfy the conditions described by these parameters:

```
char *parentWndTitle,
char *parentWndClass,
bool parentHasParent,
int parentWndX1, int parentWndY1,
int parentWndX2, int parentWndY2.
```

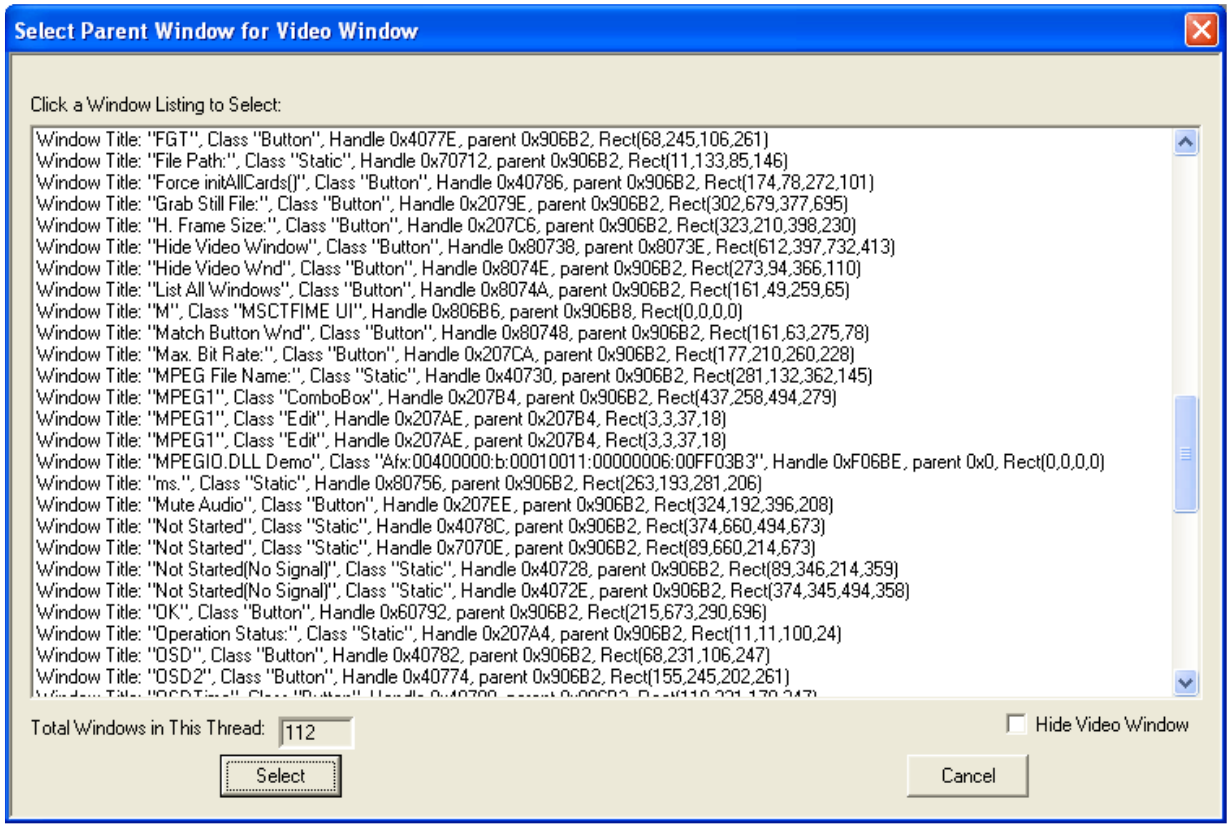
If a match is found, the found window will be used as the **ParentWindow.**
If no match was found, the current Foreground window will be used as the

ParentWindow. If no Foreground window exists then MPEGIO_start() fails.

During the search for a matching **ParentWindow**, if a char * value supplied is NULL , or an int value supplied is –1, then that parameter will not be used as matching parameter, i.e., any value from a window will match.

notReadInitFileAtStart: TRUE will cause MPEGIO.DLL to start using all default values as listed in the "Default Parameter Values" section of the <<MPEGIO User Manual>> , FALSE will cause MPEGIO.DLL to start using values in the **SYSTEMDRIVE\ MPEGIO.INI** file, where "SYSTEMDRIVE" is content of the PC's environment variable "SYSTEMDRIVE", usually this is C:\. Each time on exit, MPEGIO.DLL will automatically save its current settings (window positions, encoding parameters, etc) into this **MPEGIO.INI** file.

ListAllChildWnd: non-zero will start the "**Select Parent Window for Video Window**" dialog (suitable as a debugging tool to identify a window as **ParentWindow**):



If **ListAllChildWnd == 1**, this dialog lists all windows belonging to the **current Thread** from where the **MPEGIO_start**() has been called within an active window, i.e., somewhere user clicked mouse or pressed keyboard from one of your application software's windows.
See the "List Thread Wins" check box example in the C++ Sample source code.
If **ListAllChildWnd == 2**, this dialog lists all windows existing in the current operating system, i.e., all windows on the PC. See the "List All Wins" check box in the C++ Sample code.
If **ListAllChildWnd == 3,** No dialog will appear, MPEGIO video windows will be created as top-level windows without parent: they can be resized, dragged and moved around using mouse.
If **ListAllChildWnd == 0**, a search for **ParentWindow** will start, as described previously**,** this is demonstrated in the "Match ButtonWnd" check box in the C++ Sample source code.
Note if you pass non-zero to ListAllChildWnd, you must pass NULL to ParentWindow.

The windows listed in the "**Select Parent Window for Video Window**" dialog are represented by their title, class name, handle, parent window handle, and rectangle coordinates.
By mouse clicking/highlighting one window item then click the "**Select**" button, MPEGIO_start() will use the selected window as the **ParentWindow.**

If the box "**Hide Video Window**" is ticked then the video window will be hidden after the call to MPEGIO_start() succeeds, i.e., no video will be displayed live.

ParentWndTitle: Title (text) of the window to be searched as **ParentWindow.**
ParentWndClass: Class Name of the window to be searched as **ParentWindow.**
ParentHasParent: If the window to be searched as **ParentWindow** has parent.
parentWndX1, **parentWndY1**: The upper-left corner's (X,Y) co-ordinates of the window to be searched as **ParentWindow,** as client co-ordinate relative to its parent window.
ParentWndX2, **parentWndY2**: The lower-right corner's (X,Y) co-ordinates of the window to be searched as **ParentWindow,** as client co-ordinate relative to its parent window.

ParentWndHide: TRUE will hide the video window (no video displayed even though call to MPEGIO_start() succeeds).

Note:
(1). After successfully calling **MPEGIO_start()** on a MPEGIO card, calling this function the second time with the same BoardNumber simply receives the currently running MPEGIO card's Video Window handle.
(2) The "**current Thread**" is the Windows' thread the application software calling MPEGIO_start() belongs to, therefore the windows listed by the dialog "Select Parent Window for Video Window" when **listAllChildWnd** is 1, and the windows searched for **ParentWindow** by the MPEGIO_start() when parameter **listAllChildWnd** is 0, all belong to the current application's thread. The result is, when **listAllChildWnd is < 2,** no window outside the user's application software will be used as **ParentWindow**. In comparison, when **listAllChildWnd is 2,** any window, even a window created by a software totally un-related to your application software, can be used as **ParentWindow** (see Note **(3)** for more details).
(3) When **listAllChildWnd == 2,** depending on PC's speed**,** it will take sometime before the dialog window "Select Parent Window for Video Window" appears on screen. If a window outside your application software is selected as **ParentWindow,** that window is responsible for resizing the video window inside it: if it does not resize the video window as its size changes, the MPEGIO's video window will remain at the initial size when MPEGIO_start() was called.
(4) If MPEGIO_start() is called without supplying full list of its parameters**,** all parameters from the first missing one will use their default values as in the definition of MPEGIO_start(), e.g., if "**MPEGIO_start**(1, NULL, false, 0, "ButtonWnd 0")" is called, then these parameters will have their default values used:

```
char *parentWndClass = NULL,
bool parentHasParent = false,
int parentWndX1 = -1, int parentWndY1 = -1,
int parentWndX2 = -1, int parentWndY2 = -1,
bool parentWndHide = false.
```

Return: If successful the started MPEGIO card's Video Window handle, otherwise NULL.

-- HWND **MPEGIO_startQuick**(int BoardNumber , HWND ParentWindow,
bool notReadInitFileAtStart = false)

Function: Start one MPEGIO Card with valid parent window handle.
Parameters:

BoardNumber: Which MPEGIO card to start. If there is only one card in the PC, pass 0. When there are N > 1 cards installed, pass value between [0, N-1].

ParentWindow: any valid window's handle served as a parent window for this started MPEGIO card's video window (live video will appear inside **ParentWindow** if **ParentWindow** is visible). This parameter MUST be a valid window handle or the function will fail.

notReadInitFileAtStart: TRUE will cause MPEGIO.DLL to start using all default values as listed in the "Default Parameter Values" section of the <<MPEGIO User Manual>> , FALSE will cause MPEGIO.DLL to start using values in the **SYSTEMDRIVE\ MPEGIO.INI** file, where "SYSTEMDRIVE" is content of the PC's environment variable "SYSTEMDRIVE", usually this is C:\. Each time on exit, MPEGIO.DLL will automatically save its current settings (window positions, encoding parameters, etc) into this **MPEGIO.INI** file.

Return: If successful the started MPEGIO card's Video Window handle, otherwise NULL.
Note: This function is same as **MPEGIO_start()** in the previous version (prior to version 1.0.7.0) SDK.

-- bool **MPEGIO_boardStarted** (int BoardNumber)
Function: Return true if MPEGIO of BoardNumber has been started.

-- void **MPEGIO_stop**(int BoardNumber);
Function: Stop one MPEGIO card, if it has been started successfully previously.
Parameters:

BoardNumber: which MPEGIO card to stop, must be 0 ~ (Total Cards – 1).

Note: It is highly recommended to stop all encoding, decoding, transcoding, transmitting operations on the MPEGIO card before calling this function.

-- void **MPEGIO_stopAllCards**(void);
Function: Stop all MPEGIO cards.
Note: It is highly recommended to stop all encoding, decoding, transcoding, transmitting operations on all MPEGIO cards before calling this function.

-- int **MPEGIO_encodeStart**(int BoardNumber , int recordFileNaming,
char *fileName, char *filePath);
Function: Start recording incoming video into a file.
Parameters:

BoardNumber: which MPEGIO card to encode, must be 0 ~ (Total Cards – 1).

recordFileNaming: **0 -- prompt file name before recording**
1 -- prompt file name after recording
2 – use pre-defined file name in fileName & filePath

fileName: The non-directory part of the recording file name path

filePath: The directory part of the recording file path, inc. the '\', e.g. c:\

Return Value: SUCCESS(0) if successful, FAILURE(-1) if failed.
Note: The **fileName** and **filePath** are only used when **recordingFileNaming** is 2.

-- void **MPEGIO_encodeStop**(int BoardNumber);
Function: Stop recording on one MPEGIO card.
Parameters:
BoardNumber: which card to stop encoding, must be 0 ~ (Total Cards – 1).

-- void **MPEGIO_encodeSplitFile**(int BoardNumber);
Function: During the Record mode, start the subsequent recording data into a new file and close the current recording file. The new file will be automatically named after the very first recording file with a 4-digit serial number attached to it, and this number will automatically increase by one if this function is called continuously. For example, after **MPEGIO_encodeStart**(1, 2, "tvrecord.mpg", "c:\\today\\") is called, if MPEGIO_encodeSlitFile(1) is called twice before the MPEGIO_encodeStop(1) is called, three files will contain the completely recorded video contents from the second MPEGIO card(BoardNumber 1): "c:\\today\\tvrecord.mpg", "c:\\today\\tvrecord0001.mpg", and "c:\\today\\tvrecord0002.mpg". Each of these files can be played back independently.
Parameters:
BoardNumber: which card to start splitting, must be 0 ~ (Total Cards – 1).
Note: the MPEGIO card must be in **Record** mode for this function to succeed

-- void **MPEGIO_encodeSplitFileSize**(int BoardNumber , int InMB);
Function: During recording, create a new file each time the current file size reaches InMB megabytes.
Parameters:
BoardNumber: which card to set splitting size, must be 0 ~ (Total Cards – 1).
InMB: is mega byte unit.
Note: If InMB is 0 then no file-size based split will happen during recording for this card.

-- void **MPEGIO_encodeSplitFileTime**(int BoardNumber , int InMin);
Function: During recording, create a new file every InMin minutes.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
InMin: is minute unit.
Note: If InMin is 0 then no time-based split will happen during recording for this card.

-- void **MPEGIO_setEncodeTimer**(int BoardNumber, int minutes);
Function:Set a new timer for recording. Set the "minutes" to zero means endless recording until **MPEGIO_encodeStop()** is called or MPEGIO.DLL exits, resulted either from calling **MPEGIO_stop()** or exiting the application software.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
minutes: the new timer length. If this is shorter than the already being recorded time (since you can call this function during the recording mode), no change will happen to the timer.

-- int **MPEGIO_getEncodeTimer**(int BoardNumber);
Function: returns the currently set recording timer.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- bool **MPEGIO_getMPEGStreamType**(int BoardNumber, unsigned char *buffer,
unsigned int buffLen, int *streamType);

Function: Returns the MPEG stream type of the data stream.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

buffer: data stream content

buffLen: number of bytes in the stream pointed to by buffer

streamType: will point to value of the stream type if this call is successful. The stream types are:

0: MPEG4 VIDEO Elementary Stream
1: MPEG4 TRANSPORT STREAM,
2: MPEG4 PROGRAM STREAM,
3: MPEG2 TRANSPORT STREAM,
4: MPEG2 PROGRAM STREAM,
5: MPEG1 SYSTEM STREAM

Return Value: True(non-zero) if successful, False(zero) if failed.

-- bool **MPEGIO_getMPEGFileType**(int BoardNumber, char *fileName, int *fileType);

Function: Returns the MPEG file type of the file "fileName".

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

fileName: The mepg file name inc. full path

fileType: point to the MPEG file type if this call is successful.
The returned file type is the same as the "streamType" listed in the previous function "**MPEGIO_getMPEGStreamType()".**

Return Value: True(non-zero) if successful, False(zero) if failed

-- void **MPEGIO_setEncodeCBC**(int BoardNumber, MPEGIO_callback_get_streamC f);
-- void **MPEGIO_setEncodeCBS**(int BoardNumber, MPEGIO_callback_get_streamS f);

Function: Set up a callback function for **Record**, **Transmit** or **Transcode** operation. The user-defined function "f" which can be of any name, needs to be defined as
int f(LPSTR buffer, unsigned int length).
Once **MPEGIO_setEncodeCBC** or **MPEGIO_setEncodeCBS** is called, each recording or transmitting mode will see this "f" being called repeatedly by the MPEGIO system whenever there are encoded video data arriving from the MPEGIO card. The number of bytes carried in "buffer" is indicated by the value of "length". The user application software is responsible for immediate copying or using these data bytes, or they will be replaced by the next arriving data very soon.

The difference between **MPEGIO_setEncodeCBC** and **MPEGIO_setEncodeCBS** Is highlighted in the different definitions of MPEGIO_callback_get_streamC and MPEGIO_callback_get_streamS:

typedef int (* **MPEGIO_callback_get_streamC**)(LPSTR b, unsigned int length);
typedef int (__stdcall * **MPEGIO_callback_get_streamS**)(LPSTR b, unsigned int length);

The **MPEGIO_setEncodeCBC** and **MPEGIO_callback_get_streamC** combination is used by languages such as C++, that require the function caller to clear the stack after function calls. The **MPEGIO_setEncodeCBS** and **MPEGIO_callback_get_streamS** combination is used by languages such as Visual Basic that require the function callee to clear the stack before function call ends.

Setting “f” to null then call this function will cancel the callback during record and transmit operation: this is same as calling **MPEGIO_clearEncodeCallback().**

If **MPEGIO_setEncodeCBC** or **MPEGIO_setEncodeCBS** has never been called then the record/transmit/transcode operation will never call any callback function.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Note: The user-defined function “f” should always return quickly and avoid lengthy processing, or the MPEGIO.DLL system could fail.

-- void **MPEGIO_clearEncodeCallback**(int BoardNumber);

Function: Cancel the callback function set up previously by the **MPEGIO_setEncodeCBC** or **MPEGIO_setEncodeCBS**, the result is the recording and transmitting mode will not call any callback function anymore.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_setEncodeStopAtSignalLoss**(int BoardNumber, int seconds);

Function: Setup the time of video signal loss before Recording Stops

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

seconds:

If seconds > 0 then record will stop when signal loss has happened for these seconds of time.
If seconds <= 0 the record will not stop until MPEGIO_encodeStop() is called. By default(if MPEGIO_setEncodeStopAtSignalLoss() is never called) record will not stop until MPEGIO_encodeStop() is called by the application software.

-- void **MPEGIO_setEncodingTVFormat**(int BoardNumber, int S);

Function: Setup Recording TV Format (PAL or NTSC)

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

S: = 1 means PAL, = 2 means NTSC

-- int **MPEGIO_getEncodingTVFormat**(int BoardNumber);

Function: Returns the Recording TV Format (1 = PAL, 2 = NTSC)

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_setVideoSource**(int BoardNumber, int S);

Function: Select the video input source

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

S: =0: Composite, =2: Svideo. Other values will be ignored.

-- int **MPEGIO_getVideoSource**(int BoardNumber);

Function: Returns the current video input source: 0: Composite, 2: Svideo

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_setEncodingType**(int BoardNumber, int **t**);
Function: Set the encoding(recording) video type.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
t: 0:DVD; 1: MPEG1; 2:MPEG2; 3:SVCD; 4:MPEG4; 5:VCD
Return 0 if successful, -1 if fails

-- int **MPEGIO_getEncodingType**(int BoardNumber);
Function: Returns the encodingType:
0:DVD; 1: MPEG1, 2:MPEG2; 3:SVCD; 4:MPEG4;5:VCD
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_setEncodingHFrameSize**(int BoardNumber, int S);
Function: Set encoding horizontal frame size
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
S:176, 352, 480,528,544,640,704,720. These are used for the horizontal frame size. The valid vertical frame size, although can be set by calling **MPEGIO_setEncodingVFrameSize,** actually depends on the encoding TV system is PAL or NTSC. In PAL, the valid Horizontal X Vertical encoding frame size combinations will be 176X144, 176X576, 352X288, 352X576, 480X576, 528X288, 544X288,640X288,640X576,704X576, 720X576 pixels, etc. In NTSC, the valid encoding frame sizes will be 176X120, 176X480, 352X240, 352X480, 480X480, 528X240, 544X240, 640X240, 640X480,704X480, 720X480 pixels, etc. Depending on encoding type, some combinations of horizontal X vertical frame size might not be successful.
Return SUCCESS(0) or FAILURE(-1)
Note: MPEG1/VCD encoding type can only allow 352X288(PAL) or 352X240(NTSC) resolution.

-- int **MPEGIO_getEncodingHFrameSize**(int BoardNumber);
Function: Returns one of 176/352/480/528/544/640/704/720 as horizontal video encoding frame size.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_setEncodingVFrameSize**(int BoardNumber, int S);
Function: Set video encoding vertical frame size
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
S: for PAL: one of 144,288,576. for NTSC: one of 120, 240, 480. These are used for the vertical frame size. The valid horizontal frame size, although can be set by calling function **MPEGIO_setEncodingHFrameSize,** actually depends on the encoding TV system is PAL or NTSC. In PAL, the valid Horizontal X Vertical encoding frame size combinations will be 176X144, 176X576, 352X288, 352X576, 480X576, 528X288, 544X288, 640X288, 640X576, 704X576, 720X576 pixels, etc. In NTSC, the valid encoding frame sizes will be 176X120, 176X480, 352X240, 352X480, 480X480, 528X240,544X240,640X240,640X480,704X480, 720X480, etc. Depending on encoding type, some combinations of horizontal X vertical frame sizes might not be successful.
Return SUCCESS(0) or FAILURE(-1)

-- int **MPEGIO_getEncodingVFrameSize**(int BoardNumber);
Function: Return one of 120/144/240/288/480/576 as vertical video encoding frame size.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_setEncodingBitRate**(int BoardNumber, int R);
Function: Set up the maximum video encoding bit rate
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
R: must be between[50, 15000], unit is Kbps.
Return SUCCESS(0) or FAILURE(-1)
Note: Depending on encoding **Frame-Size** used, **MPEGIO** hardware has the following
Minimum Encoding Bit Rates:
128**K**bps for 176X144(120)-Pixel Frame Size Encoding
512**K**bps for 352X288(240)-Pixel Frame Size Encoding
1.5**M**bps for 480X576(480)-Pixel and above Frame Size Encoding

While the **MPEGIO** software allows encoding below these Minimum Bit Rates, there is no guarantee that video files encoded below the minimum bit rates can always play back well: the general rule is MPEG4 video can be encoded at much lower bit rates than MPEG1/MPEG2 video while still being played back OK.

-- int **MPEGIO_getEncodingBitRate**(int BoardNumber);
Function: Returns the maximum video encoding bit rate in Kbps unit.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_ setEncodingBitRateAverage** (int BoardNumber, int b);
Function: Set up average video encoding bit rate
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
b: must be between[50, 15000], unit is Kbps.
Return SUCCESS(0) or FAILURE(-1)
Note: 1. This function is only meaningful when **VBR** is enabled (see **MPEGIO_setEncodingCBR**())
2. See **Note** under **"MPEGIO_setEncodingBitRate()"** for **Minimum Encoding Bit Rates**

-- int **MPEGIO_getEncodingBitRateAverage** (int BoardNumber);
Function: Returns the average video encoding bit rate in Kbps unit.
Parameter:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_setEncodingAspectRatio**(int BoardNumber, int A);
Function: Set encoding aspect ratio
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
A: must be 1 for Square PEL, 2 for 4:3, 3 for 16:9. Other values ignored .
Return 0 if succeful, -1 if fails

-- int **MPEGIO_getEncodingAspectRatio**(int BoardNumber);
Function: Return encoding aspect ratio(1/2/3), see **MPEGIO_setEncodingAspectRatio**
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
-- int **MPEGIO_setEncodingCBR**(int BoardNumber, int cbr)
Function: Set encoding CBR(Constant Bit Rate) or VBR(Variable Bit Rate)
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
cbr: 1 for CBR, 0 for VBR
Return: SUCCESS(0) or FAILURE(-1)

-- int **MPEGIO_getEncodingCBR**(int BoardNumber);
Function: Get encoding CBR(Constant Bit Rate) or VBR(Variable Bit Rate)
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Return: 1 for CBR, 0 for VBR

-- int **MPEGIO_setEncodingBadSyncDetect**(int BoardNumber, int bs)
Function: Enable or disable Bad-Sync Detection during video encoding
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
bs: 0 for disable(default), non-0 for enable Bad-Sync Detection during encoding
Return: SUCCESS(0) or FAILURE(-1)
Note: This function is only valid when the encoding stream type is Program Stream.

-- int **MPEGIO_getEncodingBadSyncDetect**(int BoardNumber);
Function: Get enabling/disabling status for video Bad-Sync Detection during encoding
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Return: 1 is enabling, 0 is disabling(default).
Note: This function is only valid when the encoding stream type is Program Stream.

-- int **MPEGIO_setEncodingAudioParameters**(int BoardNumber, int format,
int samplingRate, int bitRate);
Function: Set encoding audio parameters.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
format: 0 for AC3, 1 for MPEG1 Layer 2
samplingRate: 1:32KHz, 2: 44.1KHz, 3: 48KHz
bitRate: one of 32000, 48000, 56000, 64000, 80000, 96000, 112000, 128000,
160000, 192000, 224000, 256000, 320000, 384000, in bps.
Returns: SUCCESS(0) or FAILURE(-1)

-- void **MPEGIO_getEncodingAudioParameters**
(int BoardNumber, int *format, int *samplingRate, int *bitRate);
Function: Returns the values as set in the **MPEGIO_setEncodingAudioParameters()**
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
int *format, int *samplingRate, int *bitRate:
addresses of the variables to hold the returned
values as described in **MPEGIO_setEncodingAudioParameters**

-- bool **MPEGIO_playAFile**(int BoardNumber, char *filename, short Loop);
Function: Play video file indicated by full path “filename”.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Loop: none-zero value indicates repeated playing.

Return: True(non-zero) for success.

-- bool **MPEGIO_playMPEGDataC** (int BoardNumber, MPEGIO_callback_put_streamC fC, int MPEGStreamType);

-- bool **MPEGIO_playMPEGDataS** (int BoardNumber, MPEGIO_callback_put_streamS fS, int MPEGStreamType);

Function: Decode MPEG video data returned from application-supplied function **fC** or **fS**.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

fC: caller-supplied __cdel styled function declared as(see note 2 for more details):

```
typedef int (* MPEGIO_callback_put_streamC)(LPSTR b, unsigned int length);
```

fS: caller-supplied __stdcall styled function declared as(see note 2 for more details):

```
typedef int (__stdcall * MPEGIO_callback_put_streamS)(LPSTR b, unsigned int length);
```

MPEGStreamType: One of the following values indicating the type of MPEG stream returned from function **fC** or **fS**:

0: MPEG4 Elementary Stream
1: MPEG4 Program Stream
2: MPEG4 Transport Stream
3: MPEG2 Program Stream
4: MPEG2 Transport Stream
5: MPEG1 System Stream

Return: True(non-zero) for success.

Note: **1.** These two functions retrieve MPEG data from an application-supplied function (fC or fS) by repeatedly calling one of these two functions, then sending the retrieved data to **MPEGIO** hardware for decoding: the decoded video will appear on PC screen inside **MPEGIO**'s video preview window, as well as appear on the output SVideo/RCA socket on the back-panel of the **MPEGIO** card. The length of the retrieved MPEG data in bytes must be indicated in the returned value of **fC** or **fC**: if the returned value is zero, the **SDK** will stop its MPEG decoding process.

2. The **fC** is __cdel declared, which is used in programming languages such as C++, C, etc. and **fS** is__stdcall declared, which is used in programming languages such as VB etc. Both functions treat their parameter "**b**" as a pointer to caller (**MPEGIO-SDK**) supplied buffer area which has a maximum length in bytes as indicated by parameter "**length**". Both functions must use their return value to indicate to **MPEGIO SDK** the number of MPEG data bytes they wish to decode, and these data bytes must be put into "**b**" pointed memory area before **fC** or **fS** return: if **fC** or **fS** return zero the **MPEGIO SDK** will stop decoding process immediately.

3. Decoded audio (if present in the retrieved data stream) can be heard on PC's speakers and speakers connected to **MPEGIO**'s audio output socket (on the card's back-panel).

4. Once the decoding is started, the returned MPEG stream type as indicated by parameter **MPEGStreamType** should not change until **MPEGIO_stopPlay** is called and another decoding process is started.

-- void **MPEGIO_startPlay**(int BoardNumber);

Function: Start "select play files" dialog window. After selecting one or more video files there, clicking OK button will start playing these video files on the Video Window, as well as on the TV monitor, if any, connected to the output port of the MPEGIO PCI card.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_stopPlay**(int BoardNumber);

Function: Stop MPEG decoding process started by previous calls to **MPEGIO_playAFile**, **MPEGIO_startPlay, MPEGIO_playMPEGDataC** or **MPEGIO_playMPEGDataS**.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_setDisplayAspectRatio**(int BoardNumber, int as);

Function: Setup the Video Window's aspect ratio

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

as: 0-free, 1-4:3, 2-16:9

-- int **MPEGIO_getDisplayAspectRatio**(int BoardNumber);

Function: Returns the value set by **MPEGIO_setDisplayAspectRatio**()

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_setDisplayATVFormat**(int BoardNumber, int f);

Function: Set display TV format to f = 1 for PAL, f = 2 for NTSC

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

f: 1 for PAL, 2 for NTSC

-- int **MPEGIO_getDisplayATVFormat**(int BoardNumber);

Return: 1 for PAL, 2 for NTSC

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- int **MPEGIO_transmitStart**(int BoardNumber);

Function: Start **Transmit Mode**. In this mode, the **MPEGIO** card constantly encodes the incoming video signal into MPEG data stream, and the application-supplied callback function(installed through the MPEGIO_setEncodeCallbackFunction() call) is called repeatedly with the MPEG video data passed over as it parameters. The application software can process these passed-over video data for various applications, e.g., stream the data over a network, or write the data to files. In **Transmit Mode**, the **MPEGIO.DLL** system does not create MPEG video file, this is the major difference between the "**Record**" mode and "**Transmit**" mode.

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Return: True(non-zero) for success.

-- void **MPEGIO_transmitStop**(int BoardNumber);

Function: Stop the **Transmit Mode**, return to **Preview Mode**.

Parameter:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- bool **MPEGIO_startTranscode**(int BoardNumber, char *existingMPEGFile,
char *toBeCreatedMPEGFile, char * recordPath,

int setDecodingTVFormat);

Function: Start **Transcode Mode**: one existing MPEG file is played back and transcoded (converted) into another MPEG file with possibly different encoding parameters. These parameters can be viewed from the setup dialog invoked through the **MPEGIO_startSetupDialog**() call.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

existingMPEGFile: must be a full path including drive letter,

toBeCreatedMPEGFile: if contains only the file name(such as new.mpg") WITHOUT the path name , then the "**recordPath**"(see below) will be used as the directory portion of the file path. This parameter can also contain full path name then the "**recordPath**" will be ignored. If **toBeCreatedMPEGFile** is null or contains illegal characters the transcode operation will fail. If "**recordPath**" + **toBeCreatedMPEGFile** exists then it will be deleted before creating the new file.

recordPath: full folder name inc. the trailing '\' for "**toBeCreatedMPEGFile**"

setDecodingTVFormat: 1 for PAL, 2 for NTSC.

Return: True(non-zero) for success.

Note: **1.** the existing MPEG file must have the same TV Format(PAL/NTSC) as indicated by the **SetDecodingTVFormat** value, or the transcode will created invalid MPEG file. This means PAL file can only be transcoded into PAL MPEG file, NTSC MPEG file can only be transcoded into NTSC MPEG file, although other parameters such as data rate, frame size can be changed by the transcoding.

2. the transcoding will create invalid MPEG file if the **MPEGIO_cancelTranscode** is called, or application exits, before transcoding finishes.

-- void **MPEGIO_cancelTranscode**(int BoardNumber);

Function: Cancel transcoding. The incomplete file normally is invalid MPEG file.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- bool **MPEGIO_grabStill**(int BoardNumber , char *fileName, void *videoHeader, char *frameData, long *frameDataLen);

Function: Grab current video frame as a bitmap in "filename", also pass the image data.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

fileName: full path of the bitmap file to be created. This can NULL.

videoHeader: a pointer to "VIDEOINFOHEADER *" structure as defined in dshow.h(part of DirectShow SKD). This can be NULL

frameData: pointer to raw bitmap data. Can be NULL.

frameDataLen : byte length of **frameData**

Return: true for success.

Note: This function will fail when video preview is stopped (by calling **MPEGIO_stopVideo**).

-- int **MPEGIO_getCurrentOperationMode**(int BoardNumber, char *name = NULL);

Function: Returns the current operation mode.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

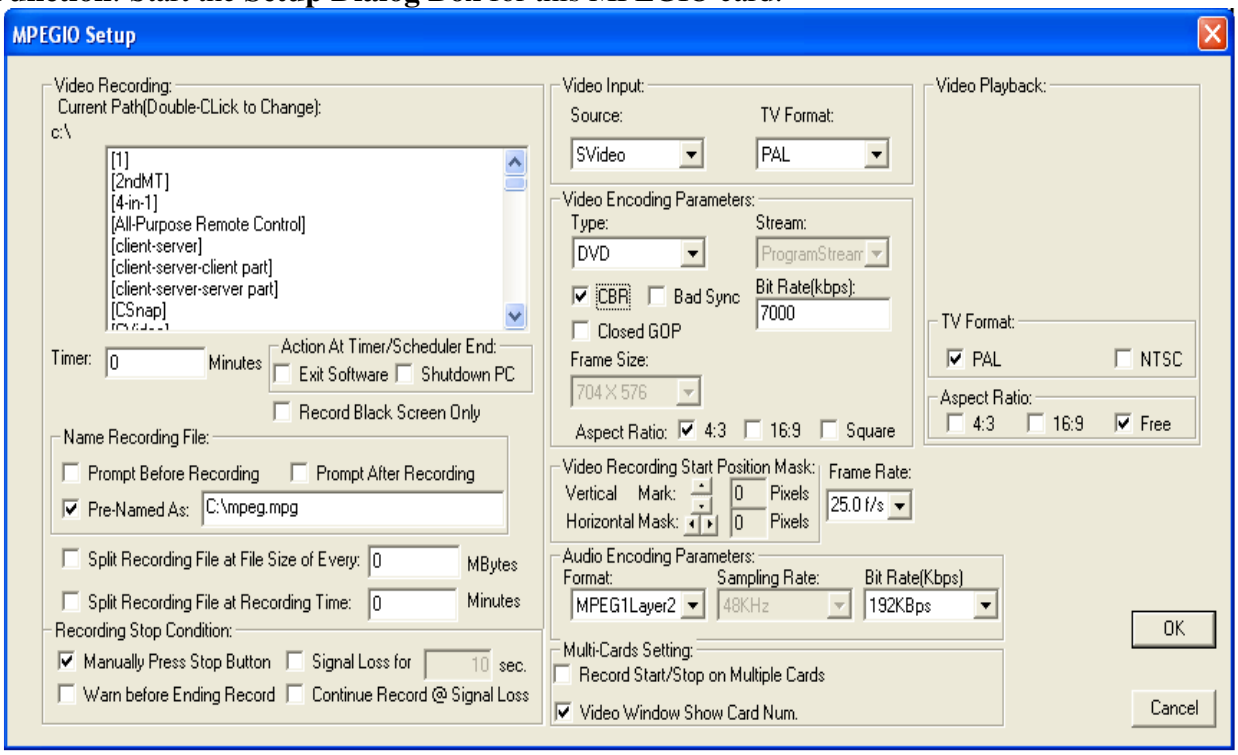
name: pointer to string to hold the returned operation name, can be NULL.

Return:

0 – preview, 1 – encode(record), 2 – decode(playback), 3 – transcode, 4 – transmit.

-- void **MPEGIO_startSetupDialog**(int BoardNumber);

Function: Start the **Setup Dialog Box** for this MPEGIO card:



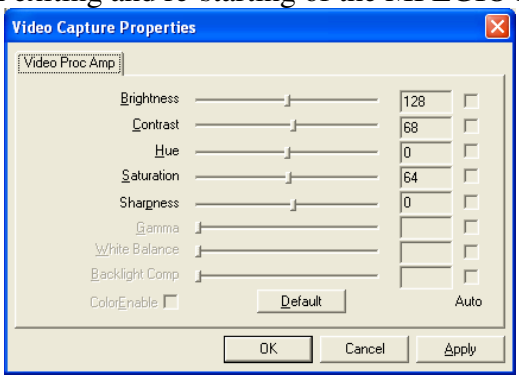
If user changed values and pressed OK the changed values will become effective.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_setVideoColour**(int BoardNumber);

Function: Start the **Colour Properties Dialog Box**. Changed colour properties will affect both encoding and decoding, and will remain effective until they are changed again, even between exiting and re-starting of the MPEGIO DLL.



Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- **bool** **MPEGIO_setVideoColourManual**(int BoardNumber, long colour, long value)

Function: Set video colour value manually: the changed colour value will affect both encoding and decoding, and will remain effective until changed again by calling this function or function **MPEGIO_setVideoColour().**

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
colour: 0 ~ 4 representing Brightness=0, Contrast=1, Hue=2, Saturation=3, Sharpness=4
value: Brightness: 0~255, Contrast: 0~127, Hue: -128 ~ 127, Saturation: 0~127, Sharpness: -8 ~ 7

Return True for success.
Note: Default colour values are: Brightness: 128, Contrast: 68, Hue: 0, Saturation: 64, Sharpness: 0

-- **bool MPEGIO_getVideoColour**(int BoardNumber, long colour, long &value)
Function: Get one encoding/decoding video colour value.
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
colour: 0 ~ 4 representing Brightness=0, Contrast=1, Hue=2, Saturation=3, Sharpness=4
value: Brightness: 0~255, Contrast: 0~127, Hue: -128 ~ 127, Saturation: 0~127, Sharpness: -8 ~ 7

Return True will see "value" be changed to the colour's corresponding value, false for failure.

-- void **MPEGIO_enableAudio**(int BoardNumber);
Function: Enable this MPEGIO card audio
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_pauseAudio**(int BoardNumber);
Function: pause this MPEGIO card audio
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- bool **MPEGIO_hasVideoSignal**(int BoardNumber);
Function: Return true if this MPEGIO card has video content being displayed
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- bool **MPEGIO_OSDBitmapLoad**(int BoardNumber, int TVFormat,
LPSTR bitmapFile, int bmpLayer, int bmpPosX, int bmpPosY);
Function: Load and show bitmap for OSD(On Screen Display)
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
TVFormat: 1 for PAL, 2 for NTSC
bitmapFile: Full path and file name of the bitmap(must be 24-bit)
bmpLayer: which bitmap(layer), must be 1 or 2
bmpPosX: upper-left horizontal position of the bitmap on the video, in pixel unit
bmpPosY: upper-left vertical position of the bitmap on the video, in pixel unit

Return: true for success, false for failure
Note: **1**.OSD appears only when the MPEGIO card is in Decode or Transcode mode
2.When both layer1 & 2 bitmap are loaded & overlapped, layer 2 bitmap is on top
3.See section "OSD(On-Screen-Display) and Live Text & Graphics Overlay" for more details

-- bool **MPEGIO_OSDBitmapClear**(int BoardNumber, int bmpLayer);

Function: Clear(remove) the OSD for this MPEGIO card/bitmap combination

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

bmpLayer: which bitmap, must be 1 or 2

Return: true for success

-- bool **MPEGIO_OSDTextDisplay**(int BoardNumber, int bmpLayer,
LPSTR text, int textX, int textY, int textW, int textH,
COLORREF textFgColour, COLORREF textBgColour, int alphaF, int alphaB,
bool TextFgTransparency, bool TextBgTransparency);

Function: Display text onto an already loaded OSD bitmap

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

bmpLayer: which bitmap, must be 1 or 2

text: NULL-terminated string containing the text to be displayed

textX: upper-left X-position of the text inside the bitmap, unit is pixel

textY: upper-left Y-position of the text inside the bitmap, unit is pixel

textW: width of each character within the "**text**" string, unit is pixel

textH: height of each character within the "**text**" string, unit is pixel

textFgColour: Foreground colour of the displayed text

textBgColour: Background colour of the displayed text

alphaF: alpha channel(blending) for foreground colour of the text, value 0~255

alphaB: alpha channel(blending) for background colour of the text, value 0~255

TextFgTransparency: if the foreground colour of the text is transparent

TextBgTransparency: if the background colour of the text is transparent

Return: true for success

-- bool **MPEGIO_OSDTextTimeDisplay**(int BoardNumber, int timeDate,
int bmpLayer, LPSTR text, int textX, int textY, int textW, int textH,
COLORREF textFgColour, COLORREF textBgColour, int alphaF, int alphaB,
bool TextFgTransparency, bool TextBgTransparency);

Function: Display text and/or time onto an already loaded OSD bitmap

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

timeDate: 1:time only, 2:time+date, -1:clear current display, 0:no time/date display

bmpLayer: which bitmap, must be 1 or 2

text: NULL-terminated string containing the text to be displayed

textX: upper-left X-position of the text inside the bitmap, unit is pixel

textY: upper-left Y-position of the text inside the bitmap, unit is pixel

textW: width of each character within the "**text**" string, unit is pixel

textH: height of each character within the "**text**" string, unit is pixel

textFgColour: Foreground colour of the displayed text

textBgColour: Background colour of the displayed text

alphaF: alpha channel(blending) for foreground colour of the text, value 0~255

alphaB: alpha channel(blending) for background colour of the text, value 0~255

TextFgTransparency: if the foreground colour of the text is transparent

TextBgTransparency: if the background colour of the text is transparent

Return: true for success

-- bool **MPEGIO_OSDIsOn**(int BoardNumber, int bmpLayer);
Function: Returns true if this MPEGIO card's bmpLayer bitmap is being displayed
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
bmpLayer: which bitmap, must be 1 or 2

-- bool **MPEGIO_OSDBitmapLoadTextTimeDisplay**(int BoardNumber,
int TVFormat, LPSTR bitmapFile, int bmpLayer, int bmpPosX, int bmpPosY,
LPSTR text, int textX, int textY, int textW, int textH,
COLORREF textFgColour, COLORREF textBgColour,
int alphaF, int alphaB,
bool TextFgTransparency, bool TextBgTransparency);
Function: Combine the functions of **MPEGIO_OSDBitmapLoad() and MPEGIO_OSDTextDisplay()**

-- void **MPEGIO_SetContinueRecordAtSignalLoss**(int BoardNumber, bool yes);
Function: If "yes" is true, when video signal is lost the recording will write black frames into the recorded MPEG file, otherwise recording will pause until video signal resumes.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
yes: if write black frames into MPEG file at video signal loss time during recording

-- bool **MPEGIO_GetContinueRecordAtSignalLoss**(int BoardNumber);
Function: Return the value set by calling **MPEGIO_SetContinueRecordAtSignalLoss.** By default (**MPEGIO_SetContinueRecordAtSignalLoss** has never been called) this value is **false**.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_SetRecordBlackscreenOnly**(int BoardNumber, bool yes);
Function: If "yes" is true, recording will write all black video frames into the recorded MPEG file: playing back the recorded file will see only black frames, plus text/graphics overlay if they are present.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
yes: True will write only black frames into MPEG file at recording time

-- bool **MPEGIO_GetRecordBlackscreenOnly** (int BoardNumber);
Function: Return the value set by calling **MPEGIO_ SetRecordBlackscreenOnly.** By default (**MPEGIO_ SetRecordBlackscreenOnly** has never been called) this value is **false**.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_SetRecordVerticalStartPos**(int BoardNumber, int vsp);
Function: Set up the vertical start-up pixel position from where the video signal will appear in the recorded frames, video signal above this position(i.e., pixels from line 0 to line vsp -1) will be masked out with black lines. This is useful to mask out some flickering noise at the top of the recorded video frames.
Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
vsp: from 0 to 100. Default(if this function never been called) is 0(no mask).

-- int **MPEGIO_GetRecordVerticalStartPos**(int BoardNumber);
Function: Returns the value set up by calling **MPEGIO_SetRecordVerticalStartPos**
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

-- void **MPEGIO_SetRecordHorizontalStartPos**(int BoardNumber, int hsp);
Function: Set up the horizontal start-up pixel position from where the video signal will appear one each line inside the recorded frames, video signal to the left of this position(i.e., pixel 0 to pixel hsp-1) will be masked out by black dots.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
hsp: from 0 to 100. Default(if this function never been called) is 0 (no mask).

-- int **MPEGIO_GetRecordVerticalStartPos**(int BoardNumber);
Function: Returns the value set up by calling **MPEGIO_SetRecordHorizontalStartPos**
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
-- void **MPEGIO_HardwareMuteAudio**(int BoardNumber, bool mute);
Function: Mute the audio output port on the MPEGIO card(the **green** socket) as well as audio from the PC's sound card during decoding mode.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
mute: true mute audio, false enable audio
Note: This function is only valid if called during Decode(Playback) mode

-- void **MPEGIO_setMonoAudioOutChannel**(int BoardNumber, int mode);
Function: Set up audio output mode on the MPEGIO card's audio out (**green**) socket
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
mode: 0 -- Normal stereo, output channel 1 and channel 2 audio simultaneously.
1 -- Repeat channel 1 audio on channel 2 (discard original channel 2 audio).
2 -- Repeat channel 2 audio on channel 1 (discard original channel 1 audio).
Note: This function is only valid if called during Decode(Playback) mode

-- int **MPEGIO_playPauseResume**(int BoardNumber);
Function: Pause/resume during playing MPEG file.
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Return: 0 -- Failed, MPEGIO is not in decode mode
1 -- MPEGIO card was playing MPEG file now changed into being paused
2 -- MPEGIO card was paused now changed into playing MPEG file.
Note: This function is only valid if called during Decode(Playback) mode

-- bool **MPEGIO_startVideoLoopback**(int BoardNumber,
char *recordPath, char *recordFileName, bool recordBlackFramesOnly);
Function: Start both encoding and decoding: video encoded is instantly decoded and output to the video output port on the MPEGIO card, if OSD is enabled the OSD will be output there as well.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
recordPath: recording file's path, can be NULL
recordFileName: recording file's name, can be NULL
recordBlackFramesOnly: "True" will record all black frames in the MPEG file

Return: True (succeed) or False (fail)

Note: **1.** Apart from testing MPEGIO card's encoding/decoding functions simultaneously, this function is useful to realtime output text/graphics overlaid on video to external TV connected to MPEGIO card's output socket. If a second MPEGIO card is present the first MPEGIO card's video output bearing overlaid text/graphics can be recorded to MPEG files by the second MPEGIO card by connecting the 1st MPEGIO's video output to the 2nd MPEGIO's video input.

2. If **recordPath** and **recordFileName** are valid folder (recordPath must end with '\') and valid file names then first MPEGIO card's video input is recorded as the original video without text/graphics overlay. With valid values for "**recordPath**" and "**recordFileName**" supplied, if **recordBlackFramesOnly** is also **True** then recording will write all black frames into the recorded MPEG file on the first MPEGIO card, together with OSD contents if they are present --- this will record MPEG file containing no video but OSD text/graphics on a black background. If **recordPath** and/or **recordFileName** are invalid or NULL (hence no recording can be started on the first MPEGIO card, this puts MPEGIO card into an "**internal hardware/firmware loopback**" mode), the video with OSD will appear on the first MPEGIO card's video output socket (hence on the external TV if connected there) quicker than when **recordPath** and **recordFileName** are valid (hence record is on).

3. Connecting a Television on the MPEGIO card's video output port and call this function with some OSD enabled will make MPEGIO card function as a text/graphics overlay device with OSD fully controllable by software.

4. This function can only be called during Preview mode.

5. A successful call to this function will change MPEGIO operation mode into either Encode mode (when **recordPath** and **recordFileName** are valid) or Transmit mode (**recordPath** and/or **recordFileName** are NULL or invalid).

-- void **MPEGIO_stopVideoLoopback**(int BoardNumber);

Function: Stop video loop back as started by **MPEGIO_startVideoLoopback**

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Note: This function will return MPEGIO back to Preview mode. If encoding is started by **MPEGIO_startVideoLoopback** then encoding is stopped and recorded file is closed.

-- int **MPEGIO_streamStart**(int BoardNumber, DWORD IPAddr, int port, bool multicast,
int sendBufSize = 0, int recvBufSize = 0);

Function: Start video streaming

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
IPAddr: streaming TCP/IP address in 4-Byte integer format: the left-most byte holds the first IP Byte (e.g**.** 254.255.3.11 will be represented as hex integer 0xFEFF030B).
port: streaming TCP/IP port number.
multicast: If stream in multicast mode (usually IP addresses within range 224.0.0.0~239.255.255.255 are multicast addresses): one stream sent out can be received by multiple clients on the same network.
sendBufSize: the send socket buffer size in bytes, if <= 0 the default 131072(128K) will be used.
recvBufSize: the receive socket buffer size in bytes, if <= 0 the default 131072(128K) is used.

Return: Non-zero means success.

Note: (1) To successfully start streaming**,** MPEGIO card must be in Encode, Decode, Transmit or Transcode mode, i.e., when calling this function, MPEGIO card cannot be in Preview mode, nor in "internal hardware/firmware loopback" mode (**MPEGIO_startVideoLoopback** was called with invalid recordPath or invalid recordFileName). To stream live incoming video without saving local file, call this function after calling **MPEGIO_transmitStart**.

(2) Video is streamed in raw hardware-encoded/decoded format without any encapsulation.

(3) Streaming is using **UDP protocol** in either Multicast or Unicast mode, the receiving end needs to be configured accordingly.

(4) To receive MPEGIO-streamed out video over network, free streaming software such as VideoLan vlc.exe, hardware MPEG decoders such as another MPEGIO card, or any stand-alone set-top box recognizing raw-MPEG video streams can be used.

(5) To stream recorded video file contents with OSD results, call this function after successfully calling **MPEGIO_startTranscode.**

(6) Calling this function twice with different values for **multicast** and IPAddr/port, therefore streaming simultaneously to two different addresses is NOT recommended, although possible.

-- int **MPEGIO_streamStop**(int BoardNumber);

Function: Stop video streaming

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Return: Non-zero means success

-- int **MPEGIO_streamIsOn**(int BoardNumber);

Function: Test if MPEGIO is streaming video

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Return: Non-zero means streaming is on, zero means stream not on.

-- int **MPEGIO_insertUserData**(int BoardNumber, char *data, int length, char pad);

Function: Insert "length" byte of user data (pointed to by "data") into current encoding MPEG stream.

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

data: pointer to data string to be inserted, bytes beyond "length" or 120 will be ignored

length: number of bytes pointed to by "data", must be < 121

pad: byte used to pad the data to be inserted to make the total length multiple of 4

Return: Non-zero(could be > "length" if "length" was not multiple of 4) means the number of successfully inserted bytes, zero means failure.

-- void **MPEGIO_encodePauseResume** (int BoardNumber);

Function: Toggle encoding's Pause/Resume status when the MPEGIO card is in encoding mode

Parameters:

BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)

Note: **1.** The MPEGIO card must be in encoding, transcoding or streaming mode to Pause or Resume

2. The Pause / Resume status will not change immediately on calling this function, since the SDK will wait until a proper moment to switch the hardware, so use the **MPEGIO_encodePaused**() function to test if the Pause/Resume status has changed.

3. Although **MPEGIO** hardware decoding can smoothly decode MPEG files encoded with any kind of pauses, encoded MPEG files with individual pauses that each has shorter than 30 seconds duration might not be handled properly on some third-party software MPEG decoders: they might cause the playback software to pause the "paused" seconds (each is < 30 sec.) before

continuing to play the subsequent video. If frequent shorter than 30 seconds pauses need to be used during encoding, using **MPEGIO** card to transcode the resulting MPEG file to another file with the same encoding parameters will create an equivalent quality MPEG file that all software decoders can play smoothly without unnecessary pauses.

-- int **MPEGIO_encodePaused** (int BoardNumber);
Function: Test if the MPEGIO card is in encoding Pause status
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Return: 1 if the MPEGIO card is in encoding Pause status, else 0.

-- int **MPEGIO_setFrameRateCode**(int BoardNumber, int frcode);
Function: Set Video Frame Rate Code for encoding
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
frcode: must be one of the following values:
1 24000/1001(23.976) fps
2 24 fps (film)
3 25 fps (PAL) : System Default for PAL Input
4 29.97 fps (NTSC) : System Default for NTSC Input
5 30 fps (NTSC drop frame)
6 50 fps (double frame rate PAL)
7 60000/1001(59.94) fps (double frame rate NTSC)
8 60 fps (Double frame rate drop frame NTSC)
Return: 0 for success, -1 for failure

-- int **MPEGIO_getFrameRateCode**(int BoardNumber);
Function: Get Video Frame Rate Code for encoding
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
Return: 1 ~ 8 as listed under **MPEGIO_setFrameRateCode**(), or –1 for failure

-- int **MPEGIO_updateEncodingParam**(int BoardNumber, int e);
Function: Enable parameter changes set during encoding time
Parameters:
BoardNumber: which MPEGIO card, must be 0 ~ (Total Cards – 1)
e: 1 for enable, 0 for disable
Return: 0 for success
Note: Many encoding parameters set during encoding/transmitting/transcoding time need to be followed by calling this function before the parameter changes become effective. If this function is called during non-encoding time nothing will happen although it still returns 0 as success.
For example, if during encoding:
MPEGIO_SetRecordBlackscreenOnly(0, true);
MPEGIO_updateEncodingParam(0, 1);
are called, then all subsequently encoded video content on card 0 will be total blackness, until
MPEGIO_SetRecordBlackscreenOnly(0, false);
MPEGIO_updateEncodingParam(0, 1);
are called to cancel the total blackness encoding.
During video streaming, changing encoding parameters will change the streamed out MPEG video stream that will affect the remotely received video.

-- bool **MPEGIO_stopVideo**(int BoardNumber);

Function: Stop video preview of a started **MPEGIO** card, MPEG encoding functions can still be used.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1)

Return: True (succeed) or False (fail).

Note:**1.** After stopping video preview successfully, the last video frame prior to the stopping will remain being seen inside the **MPEGIO**'s video preview window until a refresh/repaint event happens on the **MPEGIO**'s video preview window.

2. After stopping video preview successfully, external application software (such as VideoLan, DataStead TVideoGrabber etc. or programs written by users themselves using DirectShow SDK) can build DirectShow filter graph on the preview-stopped **MPEGIO** card to display live video coming into **MPEGIO**'s input sockets in their own video preview window, while **MPEGIO SDK**'s mpeg encoding functions can still be used as normal --- this forms a mixed function calling scenario with external application programs using **MPEGIO**'s video preview hardware while **MPEGIO SDK** using **MPEGIO**'s MPEG encoding hardware simultaneously. Users can also build their own DirectShow filter graphs to preview **MPEGIO** video after calling this function, see section **6.5** below for more discussion on this.

3. This function can be called on a started **MPEGIO** card even during its MPEG encoding or streaming operation: MPEG video file recording or streaming will not be affected by stopping or restarting video preview.

4. External software cannot preview video on **MPEGIO** card before this function is called.

5. Calling examples to this function are in **SDK's** C++ and VisualBasic sample source codes.

-- bool **MPEGIO_restartVideo**(int BoardNumber);

Function: Resume video preview on an **MPEGIO** card that was previously stopped by calling function **MPEGIO_stopVideo.**

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1)

Return: True (succeed) or False (fail).

-- bool **MPEGIO_isVideoStopped**(int BoardNumber);

Function: Test if video preview on an **MPEGIO** card was stopped previously by calling function **MPEGIO_stopVideo.**

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1)

Return: True (succeed) or False (fail) – False includes the situation when the card is not started.

-- bool **MPEGIO_setShowBoardNum**(int BoardNumber, bool showNum);

Function: Set if to show card number inside video window when multiple **MPEGIO** are present.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

showNum: True means showing the card number, false means not showing.

Return: True (succeed) or False (fail) – False includes the situation when the card is not started.

-- bool **MPEGIO_getShowBoardNum**(int BoardNumber);

Function: Test if showing card number inside video window when multiple **MPEGIO** are present.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

Return: True (showing card number) or False – False includes the situation when the card is not started.

-- unsigned short **MPEGIO_getEncodingStreamType**(int BoardNumber);
Function: Return the encoding stream type.
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
Return: : 0 for ProgramStream or 1 for Transport Stream, -1 for error.
Note: This function is only meaningful when encoding type (see function "**MPEGIO_setEncodingType"**) is MPEG2 or MPEG4. For encoding type MPEG1/VCD, the encoding stream type is always System Stream, for encoding type DVD or SVCD the encoding stream type is always Program Stream.

-- bool **MPEGIO_setEncodingStreamType**(int BoardNumber, unsigned short newEncodingStreamType);
Function: Set the encoding stream type for MPEG2 or MPEG4 encoding.
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
newEncodingStreamType: 0 for Program Stream(default), 1 for Transport Stream.
Return: true for success, false for failure.
Note 1: When calling this function and newEncodingStreamType is different from the current Encoding Stream Type, if the **MPEGIO** card is in preview mode, the **MPEGIO** card will be reset to the new encoding stream mode (if MPEGIO is NOT in preview mode, this function fails).
2. This function is only meaningful when encoding type (see function "**MPEGIO_setEncodingType"**) is MPEG2 or MPEG4.

-- bool **MPEGIO_setEncodingVideoPID**(int BoardNumber, unsigned short newPID);
Function: Set the encoding Video PID for MPEG2 or MPEG4 Transport Stream encoding .
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
newPID: valid values are within 21 and 8190 inclusive, default Video PID is 33.
Return: true for success, false for failure.
Note: The **MPEGIO** card's encoding stream type (see function **MPEGIO_getEncodingStreamType**) must be in Transport Stream to call this function successfully.

-- bool **MPEGIO_setEncodingAudio1PID**(int BoardNumber, unsigned short newPID);
Function: Set the encoding Audio PID for MPEG2 or MPEG4 Transport Stream encoding .
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
newPID: valid values are within 21 and 8190 inclusive, default Audio PID is 34.
Return: true for success, false for failure.
Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setPCRPID**(int BoardNumber, unsigned short newPID);
Function: Set the program clock reference (PCR) PID for MPEG2 or MPEG4 Transport Stream.
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
newPID: valid values are within 21 and 8190 inclusive, default PCR PID is 33.
Return: true for success, false for failure.
Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setInitialPCR**(int BoardNumber, int newPCR);
Function: Set initial program clock reference (PCR) value for MPEG2 or MPEG4 Transport Stream.

Video PTS/DTS and audio PTS are referenced to the PCR. The time unit for PCR value is 1/45 kHz (approximately 2.22 milli-seconds).

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

newPID: valid values are 0 ~ 0xFFFF, default initial PCR value is 0.

Return: true for success, false for failure.

Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setPMTPID**(int BoardNumber, int newPMTPID);

Function: Set program_map_PID for MPEG2 or MPEG4 Transport Stream.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

newPMTPID: valid values are 21 ~ 8190, default value is 32.

Return: true for success, false for failure.

Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setTSID**(int BoardNumber, int newTSID);

Function: Set Transport Stream ID for MPEG2 or MPEG4 Transport Stream.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

newTSID: valid values are 0 ~ 0xFFFF, default value is 990 (0x3E7).

Return: true for success, false for failure.

Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setProgramNumber**(int BoardNumber, int program_number);

Function: Set the program_number field of the PAT packet. It specifies the program to which the program_map_PID is applicable. If this field is set to 0, then the following PID reference shall be the network PID. The program_number may be used as a designation for a broadcast channel.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

program_number: valid values are 0 ~ 0xFFFF, default value is 1.

Return: true for success, false for failure.

Note: The **MPEGIO** card's encoding stream type must be in Transport Stream to call this function.

-- bool **MPEGIO_setSceneChangeDetection**(int BoardNumber, int enable);

Function: enable / disable scene change detection (enabling scene detection will improve encoding quality for video containing scene changes) during encoding.

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

enable: true for enabling, false for disabling, default is disabling.

Return: true for success, false for failure.

-- bool **MPEGIO_setEncodingGOP**(int BoardNumber, int N, int M);

Function: set encoding GOP structure's N and M values**.**

Parameters:

BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);

N: valid values are 1 ~ 256, default is 15.

M: valid values are 0 ~ 3, default is 3.

Return: true for success, false for failure.

Note 1: If M is not 0, N must be a multiple of M. If M = 0, then N must be 1, this is I-Frame only encoding.

2: 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."

-- bool **MPEGIO_getEncodingGOP**(int BoardNumber, int &N, int &M);
Function: retrieve **MPEGIO** card's encoding GOP structure's N and M values**.**
Parameters:
BoardNumber: which **MPEGIO** card, must be 0 ~ (Total Cards – 1);
N: valid values are 1 ~ 256, default is 15.
M: valid values are 0 ~ 3, default is 3.
Return: true for success, false for failure.

4. OSD(On-Screen-Display) and Live Text & Graphics Overlay

4.1 The **MPEGIO** OSD is realized by the MPEG chipset with these features and limitations:

- **MPEGIO** card must be in **Decode** or **Transcode** mode to see overlaid bitmap/text
- a bitmap file must be loaded for graphics overlay(calling **MPEGIO_OSDBitmapLoad**)
- each MPEGIO card has maximum 2 bitmaps loaded anywhere on its video anytime
- when the 2 bitmaps overlap the no. 2 bitmap is always on top of the no. 1 bitmap
- text can be displayed on to loaded bitmap in multiple lines with different colours & sizes
- text is using fixed-sized font, default size is 10X18-pixel, scalable
- automatically updated time/date display is supplied as part of text display

4.2 Using two **MPEGIO** cards you can realtime record text and graphics over the incoming video, resulting in a recorded MPEG file with text/graphics overlaid on top of the video, in any one of the following two methods:

Method 1:
Start recording on the 1st MPEGIO card, then start Transcoding the currently being recorded MPEG file to another file on the 2nd MPEG card with OSD bitmap loaded/text displayed, details can be seen in the C++ sample source code. The text/graphics overlay parameters can be changed arbitrarily during the transcoding process on the 2nd MPEGIO.
Method 2:
Connect the video/audio output sockets on the 1st MPEGIO card to the video/audio input sockets on the 2nd MPEGIO card, enable OSD on the 1st MPEGIO card, then calling **MPEGIO_startVideoLoopback()** on the 1st MPEGIO card will see video/audio appear on the 2nd MPEGIO card with live text/graphics overlay, that can be recorded by the 2nd MPEGIO card in realtime.

Using **Method 2** and supplying invalid or NULL values to the "**recordPath**" / "**recordFileName**" parameters to **MPEGIO_startVideoLoopback()** will see video appearing quicker on the 2nd MPEGIO card, hence less time delays between the incoming video appearing in the 1st MPEGIO card's video window and the text/graphics overlaid video appearing in the 2nd MPEGIO card's video window.

4.3 Using one MPEGIO card you can record text/graphics on MPEG files in two steps: first record the MPEG file, then load bitmap/text, then transcode the recorded file to another MPEG file: during the transcoding you can control the loaded bitmap/text in different location, sizes, colours, transparencies, etc. There is no visual video quality loss between the originally recorded video and the transcoded video if their recording parameters are the same.

4.4 Too large bitmap (close to full D1 size-704X576) can cause memory problem on the encoding chipset. Also if a bitmap's position is partially outside the current video screen the entire bitmap will not be displayed. A practical maximum bitmap size is 700 X 540 pixels.

4.5. Scaling the default fonts to multiples of its original width/height (10X18) gives smoother result than random scaling: e.g. give values 30/72 to textW/textH in function **MPEGIO_OSDTextDisplay** achieves better looking text than giving them values 33/86

5. SDK Function Calling Sequences

(1). **MPEGIO_getSDKVer**(void);
MPEGIO_setSoftwareName(name);
MPEGIO_getTotalMPEGIOCards(void);
MPEGIO_boardStarted(int BoardNumber);
these functions can be called anytime anywhere, including on PCs without **MPEGIO** card.

(2). **MPEGIO_initAllCards(void):**
must be called before calling **MPEGIO_start**()/**MPEGIO_startQuick()**

(3). **MPEGIO_start(int BoardNumber, …) or MPEGIO_startQuick(int BoardNumber, …):**
must be called before all other functions for this MPEGIO card

(4). **MPEGIO_stop(int BoardNumber)/MPEGIO_stopAllCards():**
if one MPEGIO has been started, this must be called before exiting the application.

(5). **All other functions:** must be called between **MPEGIO_start** & **MPEGIO_stop.**

6. SDK Operation Requirement

6.1 To use MPEGIO SDK functions to write application software, **MPEGIO.DLL-related files**, Microsoft **DirectX SDK 9, and** Microsoft **.net FrameWork 1.1**(or above) are needed.

To run the sample programs **MPEGIO_DLLDemo.exe** or **testMPEGIODLL.exe** on the **MPEGIO** SDK Setup CD, **MPEGIO** device driver must be installed first: run "**SetupDrv.exe**" under the "**Driver**" folder from the SDK's Setup CD, or follow Section 5. Software Installation in the "**MPEGIO User Manual**". Note if you initially installed the driver from the **MPEGIO** application program's installation folder (normally C:\Program Files\Inventa\MPEGIO"), but then uninstalled the **MPEGIO** application program, and then installed **MPEGIO SDK**, you will need to run the "**SetupDrv.exe**" from the **MPEGIO SDK'**s "**Driver**" folder again, otherwise any programs written with the **MPEGIO SDK** will crash on start-up since the **MPEGIO.DLL** will still look for the drivers from the **MPEGIO** application program's installation folder.

To run the compiled VisualBasic program **testMPEGIODLL.exe** on the MPEGIO SDK Setup CD**,** MPEGIO device driver and Microsoft .NET Framework 1.1 (or above) must be installed (a copy of beta .NET Framework 2.0 is on MPEGIO SDK Setup CD as file "dotnetfx.exe").

To install **MPEGIO.DLL-related** files, run **SDKSetup.exe** from the **MPEGIO SDK** setup CD, this will copy files from Setup CD to:

Under folder indicated by environment variable “SYSTEMROOT” (normally C:\Windows):

ac3ps_codec.sre
boot.sre
g7xxps_codec.sre
mp3eps.sre
pscodec.sre
tscodec.sre
DefaultOpt.ini
DefaultOpt_ts.ini

(*.sre are firmware configuration files for MPEG encoding chipset, DefaultOpt*.ini are chipset default setup);

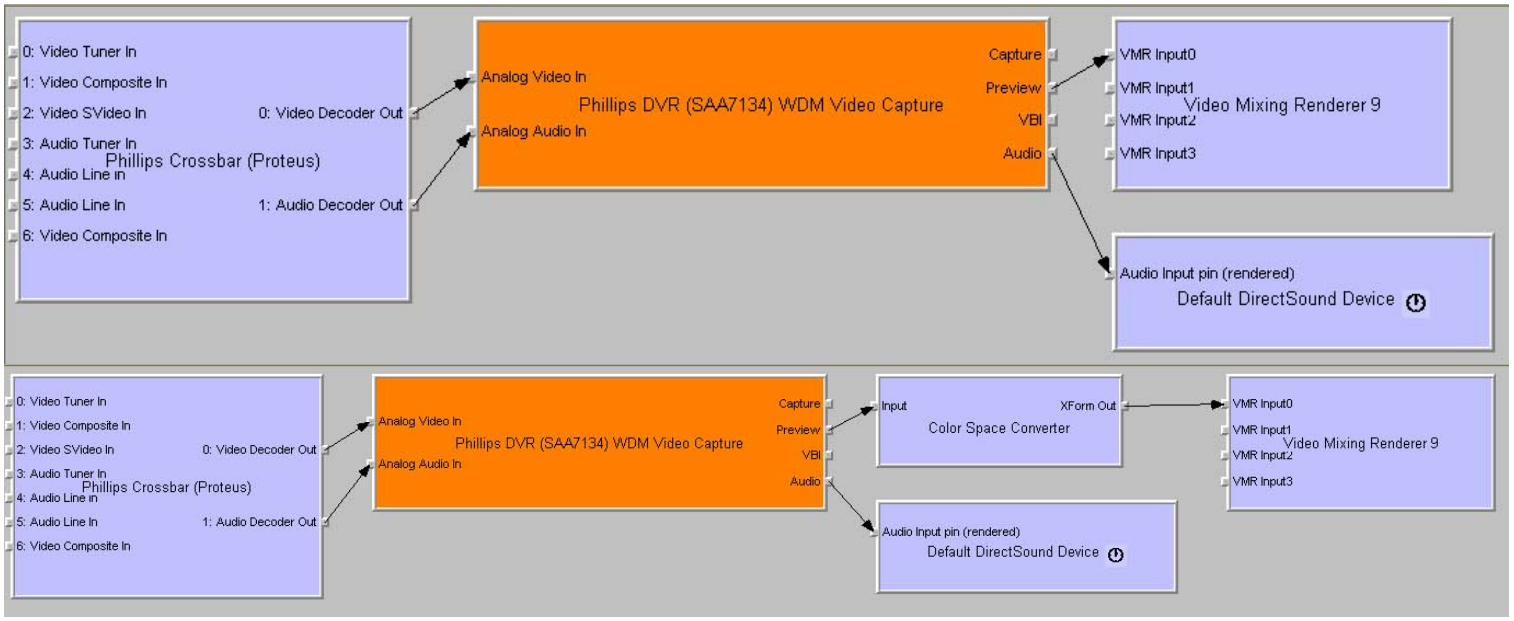
Under folder SYSTEMROOT\system32**:**

MPEGIO.dll
MPEGIO.lib
MulticastSender.dll
MulticastSender.lib
UdpSender.dll
UdpSender.lib
MPEGIO.h
MPEGIOinit.dll
VWSDKD.dll
VWSDKR.dll
DSETUP.dll
msvcp70.dll
msvcr70.dll
mfc70.dll
MPEGIODX.EXE.

6.2 To prepare a target PC to run your application software written with **MPEGIO SDK**, copy the SDK Setup CD root directory’s contents to a folder on the target PC hard disk, from there run the **SDKSetup.exe** with a command line switch /t: **“SDKSetup.exe /t”.** All the contents in the root directory (except the SDK manual) plus the “**Driver**” folder contents are needed for the target PC.

6.3 Install **Microsoft DirectX SDK Feb 2005** (dxsdk_feb2005.exe is included on the MPEGIO SDK setup CD), preferably into directory **“C:\ DirectX9 SDK” ,** assign this installation path to environment variable “DIRECTXSDK” since the sample C++/VB VisualStudio projects use this environment variable as search path.

6.4 The function prototype declaration include file **MPEGIO.H** need to be used by C++ projects, while VisualBasic projects need to declare individual MPEGIO functions one by one.
6.5 A copy of **Microsoft DirectShow SDK** is at http://inventaaustralia.zftp.com/MPEGIO/DirectShow.rar, which although is not required to develop **MPEGIO SDK** software, but will be needed in developing DirectShow-related programs, e.g. programs to build DirectShow graph to separately preview **MPEGIO** video after calling **MPEGIO_stopVideo** function to stop video preview by **MPEGIO SDK**. To build your own **MPEGIO** video preview graph the following **GraphEdit** illustrations can be used as filter connection guides:



7. Special Notes

7.1 When running software written with the **MPEGIO SDK**, the Inventa MPEGIO.exe application software distributed together with **MPEGIO** PCI card should not be running.

7.2 The Visual Basic support of calling C Functions through C .DLL is not as robust as C++ support, therefore extreme care is needed in using functions such as **MPEGIO_setEncodeCallback**(BoardNumber, **f**) where "**f**" needs to be declared as: **Function f(ByVal buffer as String, ByVal length as Integer) as Integer** in a separate Module outside the Form module. Full details are in the VB sample folder of the SDK Setup CD.

7.3 In Non-C++ programming environment, treat the bool variable type as 32-bit Integer: e.g., in VB and Java, declare function **MPEGIO_boardStarted**() as returning Int32**,** rather than Long or Boolean**,** because Long in VB and Boolean in Java are 64-bit variables while **MPEGIO.DLL** treats bool as 32-bit integer. When using Delphi, declare any bool variable used in the **MPEGIO SDK** as LongBool (32-bit).

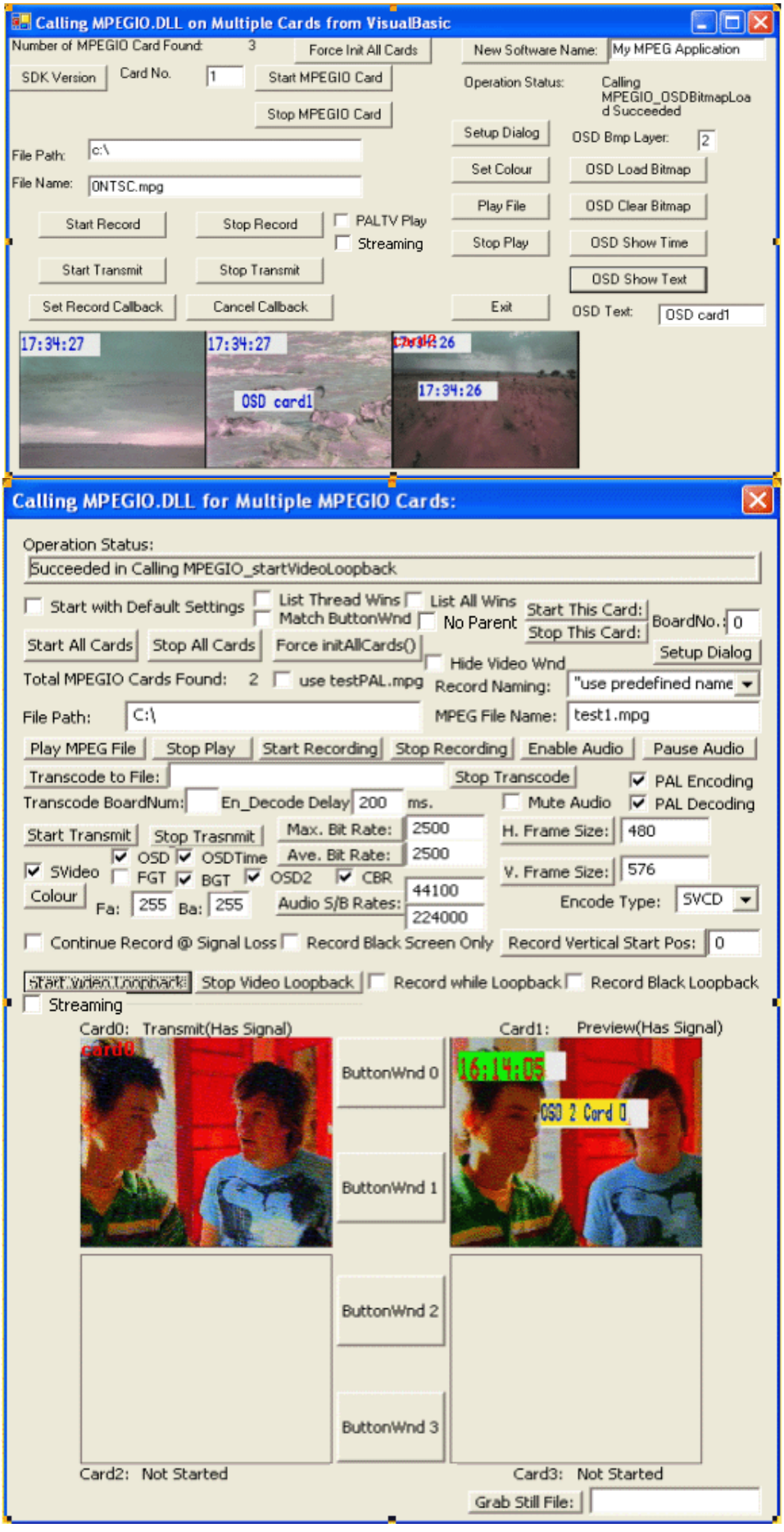
8. Sample Source Code

Two sample applications with full source codes are included on the **MPEGIO SDK** Setup CD:

MPEGIO_DLL 1.0.8.6 CPP Sample : C++ Sample

MPEGIO_DLL 1.0.8.6 VB Sample : VisualBasic Sample

Each sample has complete VisualStudio solution file (.sln) for compilation, here are some of their screen shots:


Calling MPEGIO.DLL on Multiple Cards from VisualBasic
Number of MPEGIO Card Found: 3
Force Init All Cards
New Software Name: My MPEG Application
SDK Version
Card No. 1
Start MPEGIO Card
Operation Status: Calling MPEGIO_OSDBitmapLoad Succeeded
Stop MPEGIO Card
Setup Dialog
OSD Bmp Layer: 2
File Path: c:\
Set Colour
OSD Load Bitmap
File Name: ONTSC.mpg
Play File
OSD Clear Bitmap
Start Record
Stop Record
PALTV Play
Streaming
Stop Play
OSD Show Time
Start Transmit
Stop Transmit
OSD Show Text
Set Record Callback
Cancel Callback
Exit
OSD Text: OSD card1
17:34:27
17:34:27
OSD card1
17:34:26
Calling MPEGIO.DLL for Multiple MPEGIO Cards:
Operation Status:
Succeeded in Calling MPEGIO_startVideoLoopback
Start with Default Settings
List Thread Wins
List All Wins
Match ButtonWnd
No Parent
Start This Card:
Stop This Card:
BoardNo.: 0
Start All Cards
Stop All Cards
Force initAllCards()
Setup Dialog
Hide Video Wnd
Total MPEGIO Cards Found: 2
use testPAL.mpg
Record Naming: "use predefined name
File Path: C:\
MPEG File Name: test1.mpg
Play MPEG File
Stop Play
Start Recording
Stop Recording
Enable Audio
Pause Audio
Transcode to File:
Stop Transcode
PAL Encoding
Transcode BoardNum:
En_Decode Delay 200 ms.
Mute Audio
PAL Decoding
Start Transmit
Stop Trasnmit
Max. Bit Rate: 2500
H. Frame Size: 480
OSD
OSDTime
Ave. Bit Rate: 2500
SVideo
FGT
BGT
OSD2
CBR
V. Frame Size: 576
Colour
Fa: 255
Ba: 255
Audio S/B Rates: 44100
224000
Encode Type: SVCD
Continue Record @ Signal Loss
Record Black Screen Only
Record Vertical Start Pos: 0
Start Video Loopback
Stop Video Loopback
Record while Loopback
Record Black Loopback
Streaming
Card0: Transmit(Has Signal)
Card1: Preview(Has Signal)
card0
ButtonWnd 0
ButtonWnd 1
ButtonWnd 2
ButtonWnd 3
Card2: Not Started
Card3: Not Started
Grab Still File: