UDP Dvr API 5 SDK
Developer's Guide

For Win32 Platform

API Version 5.0

UDP Technology

Contents

0. Overview

UDA5 (UDP Dvr Api ver 5) refers to interface method between UDP products operated on the PC and the application.
UDA5 SDK (Software Development Kit) refers to the environment that allows the development of software using UDA5.

Each chapter of this manual consists of the following.

In the chapter 1, SDK package and environment are explained.
In the chapter 2, UDA 5 API and software/driver structure are explained.
In the chapter 3, actual usages of the API with example source codes are demonstrated.

Please consult following manuals for further reference.

Manual Name	Description
HW Manual	HW Installation and Specification
UDA5 Reference Guide.	API reference.
SW Manual	Product specific API variation.
SampleApp User Guide	Using Sample Applications

1. About UDA5 SDK

1.1. Package detail

The SDK is distributed in a package that contains following file types.

Package	Files	Description
Driver	*.sys	Binary files for the device driver.
	*.inf	Information files for the device driver.
DLL	*.dll	Dynamic link library files
	*.lib	Library files to be linked during program compilations.
	*.h	Header files to be included in program source
Sample Program	*.exe	Executable files
	*.xml	Initialization files containing DLL / ModelID / Activation Code information. Various Configuration data.
Sample Source	*.cpp;*.h; *.dsp;*.rc…	Source files for example programs.
Manual	*.pdf	Document Files.

1.2. Runtime and compile environment

1.2.1. Runtime Environment

Hardware Requirements

Main Board	Intel BX or later chipset
CPU	Celeron/Pentium III, IV or higher CPU
Memory	256MB or more – More memory may be required due to hardware configuration
VGA	AGP or PCI-E VGA supporting Overlay function

Software Requirements

OS	Microsoft Windows 2000, Microsoft Windows XP, Microsoft Windows Vista
Other	DirectX 7.0 or higher

Even though the UDP product is actually operated in a different environment, since it is not tested sufficiently, normal operation cannot be guaranteed.

1.2.2. Build Environment

Following additional conditions are required in addition to the runtime environment

Microsoft Visual C/C++ 6.0
Service Pack 5 for Microsoft Visual C/C++ 6.0
Microsoft Platform SDK
Microsoft Direct X SDK 7.0 or higher
Processor Pack for Service Pack 5 (Optional)

- Or -

Microsoft Visual Studio.NET

2. About Version 5 API

2.1. Software System Architecture

Diagrams for the software system structure for local devices

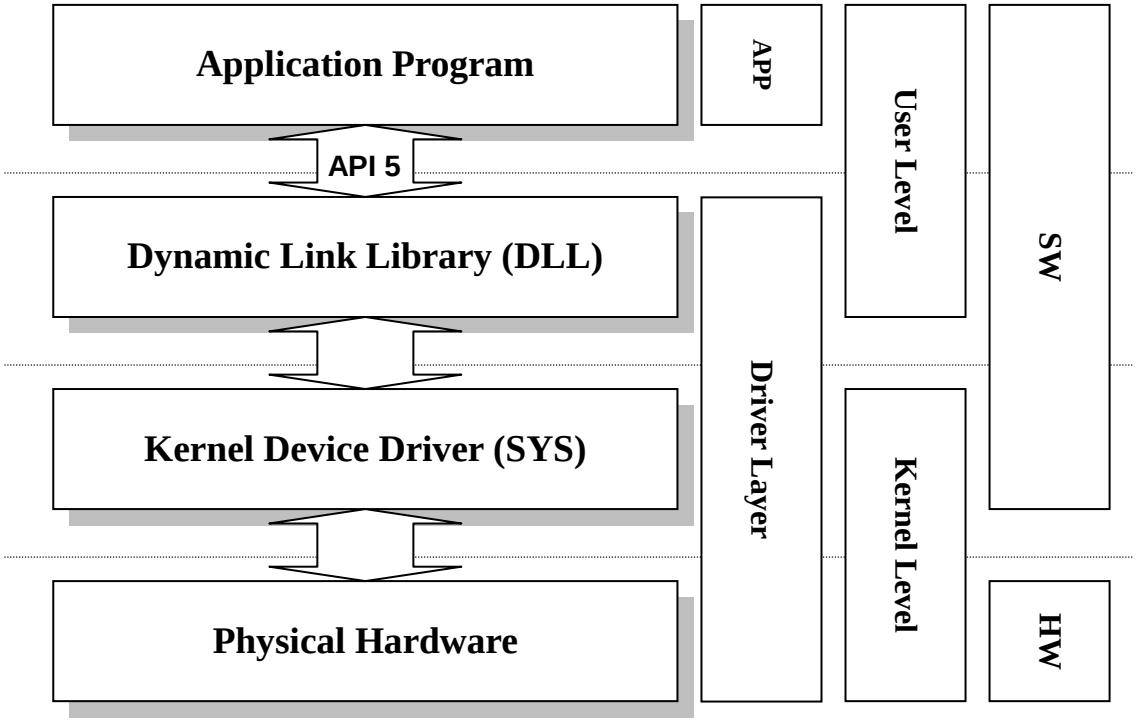
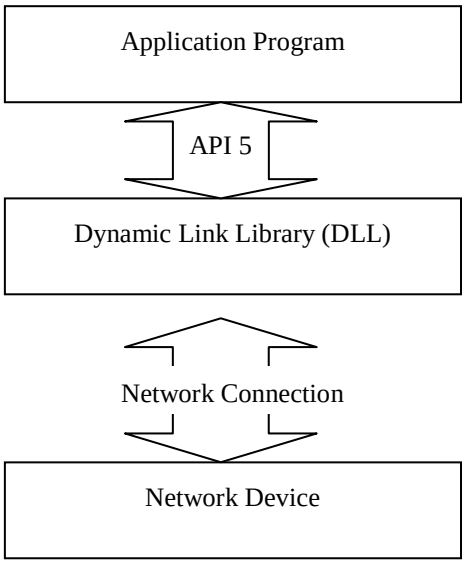


Diagram for Network device.



As shown above, the application recognizes only the Driver Layer through API 5 interface. Levels on the figure can be grouped as User / Kernel Level or SW / HW depends on a point of view. For the network device, the application program is interfaced with the device in the form of API5 in the same manner.

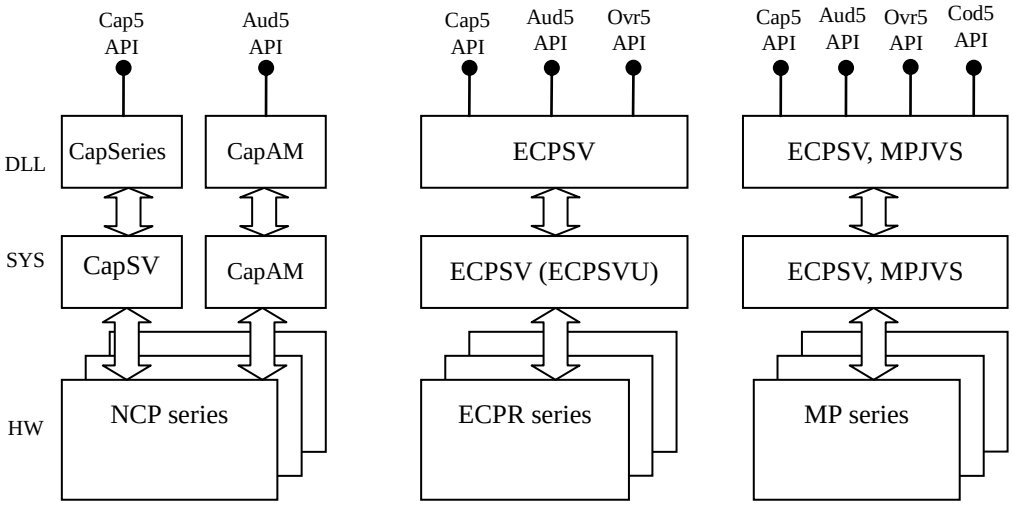
As long as the API version 5 is supported, internal changes in HW and SYS/DLL, and even the network device or local device do not have any effect on the usability of the application. This is the main reason for defining the API.

2.2. API Set Outline

The API 5 is divided into five API sets as Audio, Video, Codec, Real-time and Network. The sets are grouped by types and functions of data supported by the HW board. Descriptions and characteristics are shown below.

API Set	Function Name Type/Example	Related HW
Audio	Aud5xxx/ Aud5Setup	Generates raw audio data.
Video	Cap5xxx/ Cap5SetVideoFormat	Generates raw video data.
Codec	Cod5xxx/ Cod5ReleaseData	Generates compressed audio/video data.
RealTime	Ovr5xxx/ Ovr5Split	Transmits real-time video data to the graphic card
Network	Net5xxx/ Net5Connect	Generates various data over networks.

Each HW (Board) supports one or more of API Sets. HW supporting multiple API sets utilizes one or more of DLL/SYS pairs. Actual example figure is shown below.

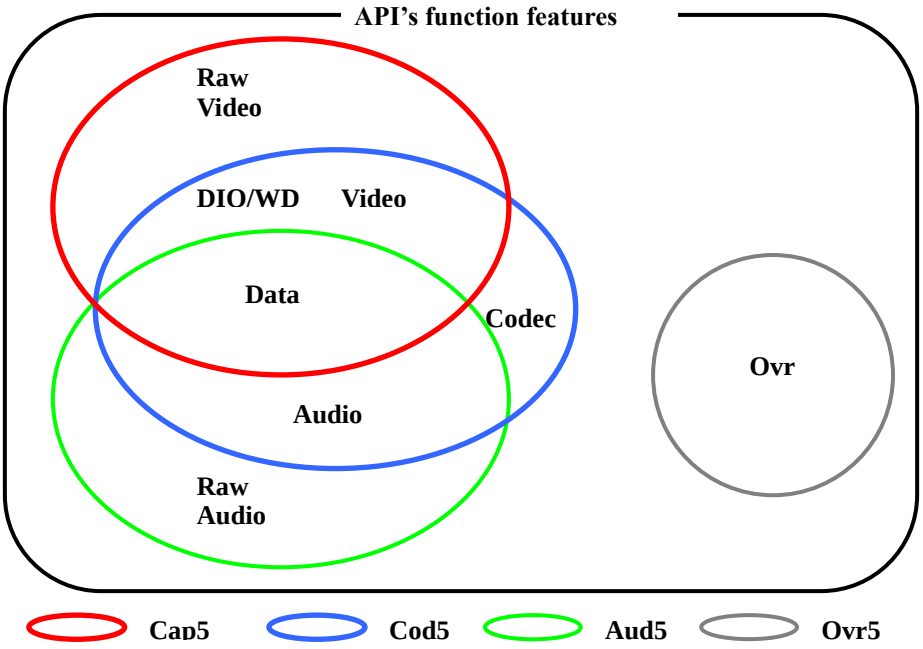


These API sets share a very similar function set(except network devices). If observed in function level, the similarity is apparent. The next chart shows relationship between the API sets and their functions.

	Common	**Data**	**DIO/ WD**	**Video**	**Audio**	**Codec**	**Raw Video**	**Raw Audio**	**Ovr**	**Net work**
Cod5	✓	✓	✓	✓	✓	✓				
Cap5	✓	✓	✓	✓			✓			
Aud5	✓	✓			✓			✓		
Ovr5	✓								✓	
Net5		✓								✓

Exclusive Function Sets for each API set are marked as Gray color.

Following is set diagram for the function sets. The Common Function Set is included in every API set.



Only the network device is involved in the network connection, etc., and it has characteristics other than the remaining 4 types of APIs.

All the APIs shown above are provided in the form of COM (Component Object Model) Interface. Only the local device is composed of double structures that support the exported function additionally.

Name and ID of Interface for Each API

API	Interface Name	Interface ID
Cap5	ICap5	IID_ICap5
Cod5	ICod5	IID_ICod5
Aud5	IAud5	IID_IAud5
Ovr5	IOvr5	IID_Ovr5
Net5	INet5	IID_Net5

All interfaces support IUnknown. The pointer for other interfaces that are supported by the corresponding device through QueryInterface can be obtained.

2.2.1. Video API Set

Functions in the set have Cap5xxx style function names.

The API set includes functions to import Raw Video type data and DIO/Watchdog function. While the API supports similar function set to Cap3 API of Suez Video, note that there are changes made.

The API set is mostly used to compress and store the Raw Data, or process and display images on the screen. However, with the CODEC / Real-time API, the Video API may substitute data compression and screen display.

The Video API set generates very large data, compared to other API sets. If used in the multi-card environment, the set generates larger data than PCI range of 133MB/s (32bit/33MHz) or PCI-E(266MB/s). To handle the large data, the APP must be carefully designed to support it.

The set supports Video Signal Format / image size (resolution) as basic video characteristics and frame rate / color format as raw video characteristics.

2.2.2. Codec API Set

The CODEC API set uses Cod5xxx format function names.

The set includes functions to load compressed Audio / Video data and DIO / Watchdog functions. The set is similar to the CAPM4 API.

Depends on the system board, the Cod5 API can generate motion data and other a variety of data format in addition to the basic compressed Audio / Video data. Also, depends on the HW board and driver, specific format of the Audio / Video stream varies.

Since the CODEC data is stream type data, a block of data such as GOP(Group of Pictures) becomes invalid when buffer overflow or data corruption occurs. When this happens, the driver must notify it to the APP and stop generating data. This step is necessary because the data might be meaningless and receiving data is not even available if there is an error in the APP itself. After the correction is made, call the ChannelEnable function from APP to restart the channel.

2.2.3. Audio API Set

The Audio API set uses SAud5xxx format function names.

The set includes functions to load Raw Audio type data. Unlike the Cod5 and Cap5 API sets, the Audio API set does not support DIO /Watchdog functions. While the API supports a similar function set to Cap3 API of Suez Video, note that there are changes made, such as removal of frame rate related functions.

2.2.4. RealTime API Set

The API set uses Ovr5xxx format function names. Unlike other API sets, the set only includes simple control functions without supporting any data. While real-time image data does not pass the CPU, it does not support data function group.

2.2.5. Network API Set

The API set uses Net5xxx format function names. Unlike other API sets, the set only includes network control functions without supporting any common features. When using the network device, only after

the connection is completed successfully using the Net5 API can the remaining API sets be used.

2.2.6. Other characteristics of the API 5

- Type API 5 supports multi-board operation. Each API function has uBDID to support the multi-board operation. Use similar or different boards to make a variety of configurations.

- All API sets include process state control functions and have own process state. Even if a board supports more than one AIP sets (MP204: Supports Cap5 and Cod5), the process control function of one API does not affect to process state of other API sets.

- In the network device, any API can be used that is supported by each of the remaining devices after Net5 has been used to get it in connection state. All connection/process control states are irrelevant to each interface.

- All API functions return a Boolean value to indicate success or failure of the function...

- List of commonly used parameters.

Parameter	Description
uBdID	Index number to the board. The nearest slot to CPU is index 0. The next is index 1, and so forth. Even if a slot is skipped, index numbering is not affected and only the order of slots is affected.
uChID	Index number to the channel. Each board in the system has index number to channel. Because actual channel index among boards might be related together, interface for all channels are not provided.
uVPID	Index number to the video processor. Each board has own index number. Used in I2C, Video FrameRate (UserSequenceList) only.
uCmd	Parameter for Property/Adjust functions. Additional parameters are used for each function.
bwData	Bitwise type data. Sensor/VideoStatus

- Each DLL supports one or more API Sets..

- The API supports parameter type functions. Depends on parameter values, one function can have multiple abilities. Suitable for additional expansion.

- API Set and Physical Device – If a function named SetBrightness is called from physically different devices, each call will be applied to it's own API set. However, when there are two API sets sharing same function name in one device, each call will affect to both API sets since they share same physical video processor. (ex: both Cap5 and Cod5 has SetAdjust – ADJ_BRIGHNESS,…) Please keep this in mind while designing APP..

2.3. API Set Features

2.3.1. Common Functions

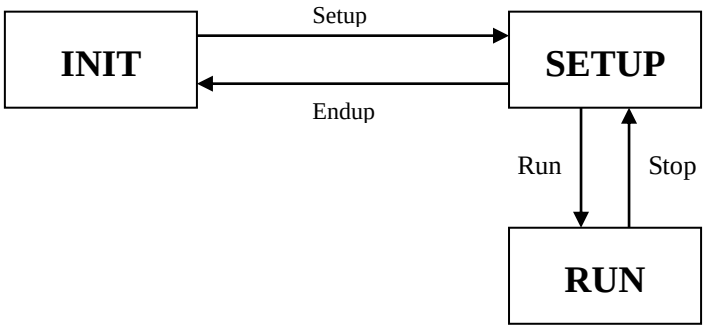
Basic function groups such as Cod5, Cap5, Aud5, and Ovr5. These groups include basic control and reporting functions.

Process Control Functions

A driver is a state machine with a static internal value. Depends on the state, available API functions are limited. A driver has one of following three states at any given time.

Status	Description
INIT	The driver has not been loaded
SETUP	The driver has initialized. The APP can set properties.
RUN	Internal routine of the driver is running. APP can run most of the API functions. The APP is processing major work.

The process control functions (Setup/Run/Stop/Endup) can change the status value. See the next figure and chart for relationship between these functions.

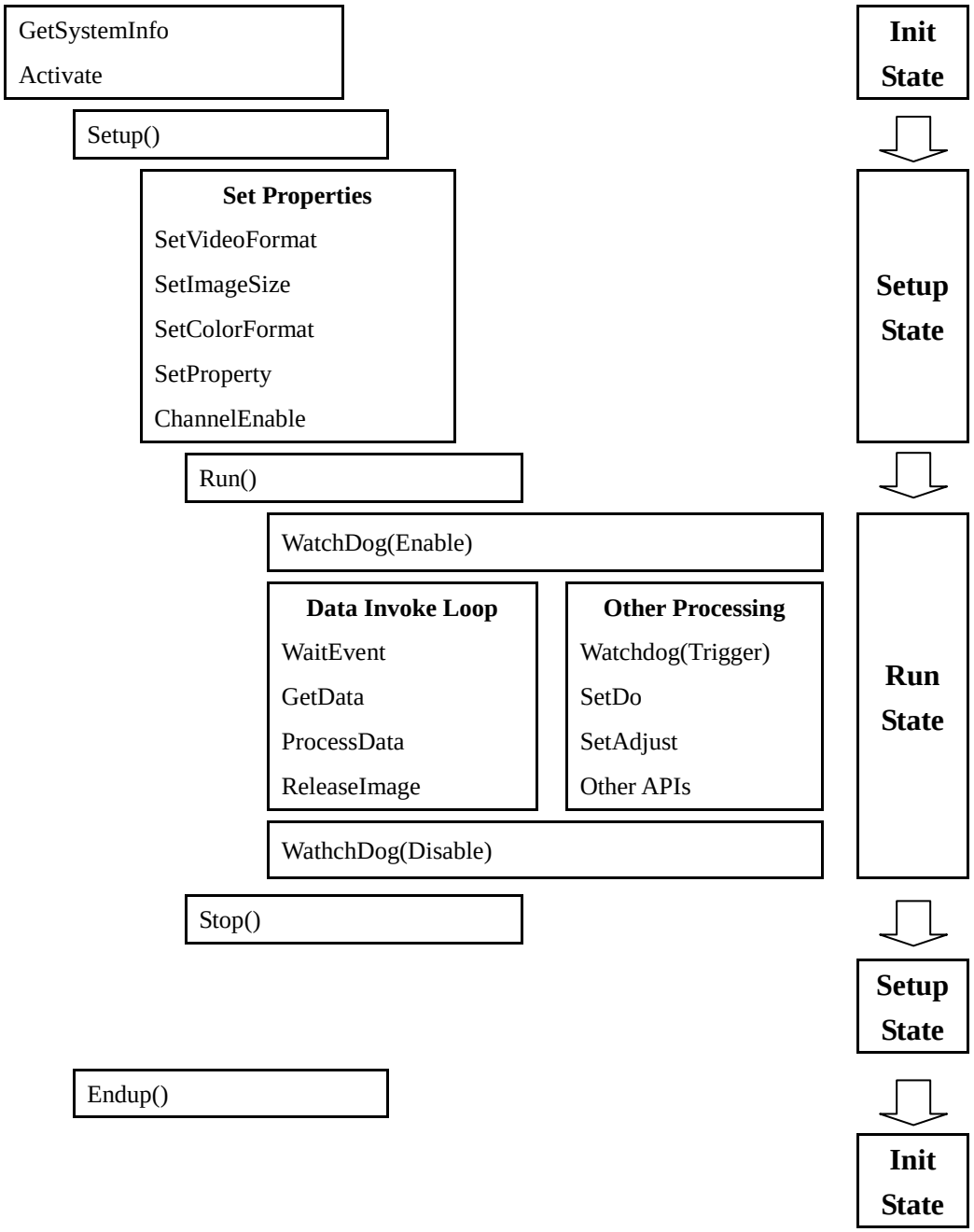


Function	Pre-Condition		Post-Condition(Succeeded)
Setup	**Init**	→	**Setup**
Run	**Setup**	→	**Run**
Stop	**Run**	→	**Setup**
Endup	**Setup/Run**	→	**Init**

Before the setup, each board needs activation to run the driver. Use Activate function of each API set to issue a Activate code. Each board model has own activation code and is distinguished by ModelID. Use BoardInfo to get the ModelID. Required Activation code fore each ID is provided separately through

Sample Code/Registry File/Ini File/Document. Every board supported by an API set need to be activated before using the API.

Below is a diagram for the relationship between ProcessControl function and other functions and the driver status.



ChannelEnable

The ChannelEnable function is used to enable/disable each channel instead of the whole API. Although the ChannelEnable function is not expressly included in the ProcessControl, it is a type of the

ProcessControl function. This function can enable or disable basic function of each channel in the board. In the Cap5, assign whether or not capture video from the channel. In the CODEC, assign whether or not make compression stream. In the Ovr, decide whether or not display the channel in the screen. Enabling this in the Setup state will affect in the next Run state. Enabling this in the Run state will affect immediately. This function is available in both the Setup and Run states.

The Process Control also decides order to use the driver API. For example, the Setup must be done before the Run. However, some API has predefined sequence of functions regardless of the Process Control. For example, some Setup state functions such as SetVideoFormat/SetImageSize/SetFrameRate have no sequence by the Process Control but have own sequence by relationship to each other.

Information Functions

All API sets in the API5 provide functions for API status and information. These functions are described below. Before the activation, use this function to decide what to activate.

- QueryInfo

Get the current system status. Mainly get the Process Control state. Check the Process Control section for the detail

- GetSystemInfo

Get number of boards and version information of the system. Receive the information in the CMN5_SYSTEM_INFO structure regardless of the API set. Usually called in the beginning of the APP

- GetBoardInfo

Get the board information. Each API has own information structure. Check CAP5_BOARD_INFO, COD5_BOARD_INFO, AUD5_BOARD_INFO, and AUD5_BOARD_INFO for each API set. Following is available information by the function.

Common values

Variable	Description
uModelID	Base value to get the Activate code for the activation. More information is available in the Process Control.
uSlotNumbe	Physical PCI slot number.
uMaxVP	Number of VP. Related to functions using the uVPID parameter.
uMaxChannel	Number of Channels. Related to functions using the uChID parameter.

- GetLastError

Structure of the API only allow returning success or failure information of the operation. Use the

GetLastError function to get detailed error information when the API has failed.
Unlike the Win32 GetLastError storing only the last error per each thread and returns only the error code, the API5 GetLastError stores error information in the internal queue and provides detailed error information and internal ErrorCode for the last and previous errors.

Property Functions

Set or get Setup state controllable values per each channel. (SetProperty, GetProperty) The Get Property is available in both the Setup and Run states but the SetProperty is available only in the Setup state. The functions use additional parameters to run the command. The Set function has four ULONG parameters and the Get function has four ULONG* parameters. Actual type and meaning of the parameters are decided by the command. Not all parameters need to be used.

Usable commands can be categorized by each function set. For example, in the Cod5SetProperty, property command of Video/Audio/CODEC functions are available. Check Audio/CODEC/Video functions and reference paragraph for the detail.

Following is example of commands supported by each function. More commands may be added in the next SDK versions.

Function	Property Commands
Audio	Sampling Frequency
Codec	Stream Type(AudVideo, Audio Only, Video Only),GOP Size
Video	Configurable items with property are already supported by functions. Therefore the Video property is not currently supported. It is still possible to add it.

Adjust Functions

Set or get parameters for each channel, which are available in the Run state. Both SetAdjust, and GetAdjust are available in both the Setup and Run states.

The functions use additional parameters to run the command. The Set function has four ULONG parameters and the Get function has four ULONG* parameters. Actual type and meaning of the parameters are decided by the command. Not all parameters need to be used.

Usable commands can be categorized by each function set. For example, in the Cod5SetProperty, property command of Video/Audio/CODEC functions are available. Check Audio/CODEC/Video

functions and reference paragraph for the detail.

Following is example of commands supported by each function. More commands may be added in the next SDK versions.

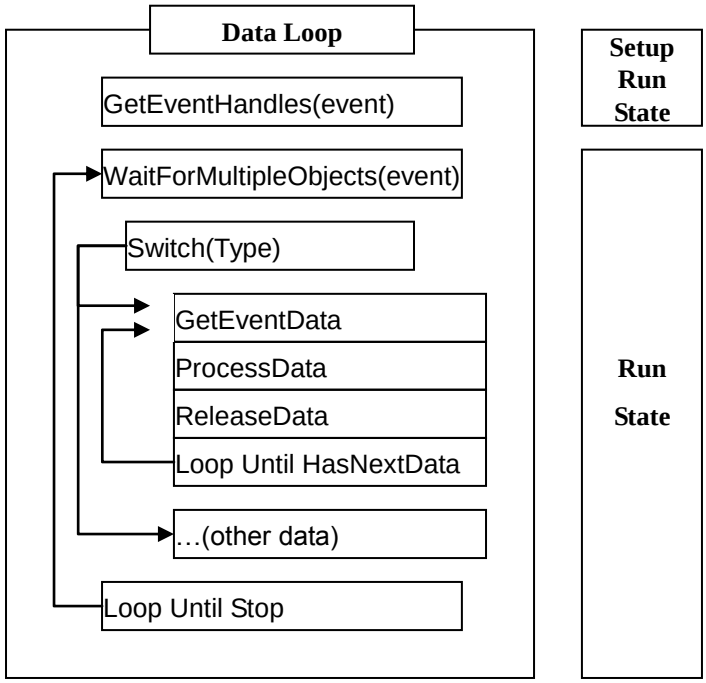
Function	Adjust Commands
Video	Brightness, Contrast, SaturationU, SaturationV, Hue
Audio	Gain
Codec	Skip Frame, Bit-rate

2.3.2. Data Functions

The Data function group provides methods to collect the board generated data. Cod5, Cap5, and Aud5 API sets support the Data function set. In the API Version 5, only Callback/Event methods are supported. Note that Query method is not supported.

Event Data Retrieve Functions

In the Event method, the GetEventHandle function is used to get event of each data type, wait for event signal, and get the type of data with the GetEventData function. Suitable if one APP supports multiple API sets simultaneously and uses only one data thread.
The next figure describes general Event method.



Each API set provides notification event of data and status through the GetEventHandle function. The type provided matches with DataType supported by each API set. Please check <Chart1-1>. After retrieving event matching each DataType, use Win32 wait function to wait for the event to check data generation of status change. If the specific data generation is sensed, call appropriate API GetEventData function to get the actual data. Use the data appropriately according to the APP (Previously done with the Callback function) and call the Release function. Check the HasNextData field to process any remaining same type of data before calling the wait function again.

Caution!
You may be confused if the released data can be used again after calling Release function. Below description might be useful for this.

1st rule : **Always call ReleaseData**.
ReleaseData should be called in any circumstances.

2nd rule : **Data validity is "unknown" just after the data has been released.**
You might have an access to the data even after you already released it. But this is beyond the UDP's principle, so it will be changed without any prior notice. Therefore, the fact that now you can access to the data even after releasing it does not guarantee that you will be able to do the same in the future. You might get a broken or null data with our future dll.

3rd rule : **Both data header and its actual data should be copied**
If you would like to have an access to the data after releasing it without any preconditions, you should copy the header information and the actual data as well (such as raw video image and compressed data). Copied data is no longer the data managed by the driver, so your APP can handle the data without any restrictions.

* Since the media data (Not status data) pointer guarantee 16byte align, use the SIMD commands in performance critical routine.

In the physically same device (For example, DI and DO in MP204 are physically same), get handle from only one API for one DataType (For example, sensor/Vstatus in Cod5 only)

Data Function's characteristic

While the callback method uses passive structure with driver control, in the Event method, the APP has control to get the data and uses own loop to repeatedly get the data. This is the Data Invoke Loop part of

the program sequence diagram.

The Event method had separate functions for data notification and data collection. Therefore, more detailed control is possible with the Event method. Regardless of the method, collected data must be released with the ReleaseData function to free up buffers for the next data.

Each API set generates the data support following data/state types. The Invoke is only available in the supported type.

	DT_VIDEO	DT_AUDIO	DT_COD	DT_SENSOR	DT_VSTATUS
Cap5	✓			✓	✓
Aud5		✓			
Cod5			✓	✓	✓

Media data such as DT_VIDEO, DT_AUDIO, and DT_COD requires corresponding ReleaseData functions. Status data such as DT_SENSOR and DT_VSTATUS do not require the ReleaseData functions. Each type has corresponding structure type as shown below

Type	Data/Status	Structure
DT_VIDEO	data	CAP5_DATA_INFO
DT_AUDIO	data	AUD5_DATA_INFO
DT_COD	data	COD5_DATA_INFO
DT_SENSOR	status	CMN5_SENSOR_STATUS_INFO
DT_VSTATUS	status	CMN5_VIDEO_STATUS_INFO

In the DT_COD type, details vary depends on characteristics of each CODEC board. Check additional manuals for the detail.

Callback methods

Can be used more easily than the event method. The callback is retrieving data through the driver's internal thread and then calling a function using the data as parameter. Since the callback function runs in ThreadContext of the APP instead of MainThread of the APP, careful synchronization for global objects are required.

StartCallback and EndCallback functions are exist to allow resource allocation and deallocation per thread. For example, the most representative function is CoInitialize()/CoUninitialize(). The data callback is called when the data is created. The App should register NULL function for the unused callback.

Type of Supporting Callback for Each API Set

API	Callback types
Cap5	Video Data, Video Status, DI Status
Cod5	Coded Data, Video Status, DI Status
Aud5	Audio Data
Ovr5	N/A
Net5	N/A

When registering the callback function, the lpcontext can be used as ID or class pointer, etc. when the callback is called.

The callback method has an advantage that it is simple but has the following restraints.

Restraints on the Callback Method

- Resolving Deadlock processing is needed.
- Adjustment of the priority of the thread is not desirous.
- Since the thread configuration cannot be expected, various safety devices are needed for the callback routine.
- Various thread configurations are impossible and it is fixed by the driver.
- Not supported in the network device.

About the occurrence of deadlock

When developed with the callback method, in some cases the stop function does not return. This phenomenon is called deadlock. Since the callback routine waits for the result from the thread that stops the function and the stop routine waits for the end of callback internally, both the threads are put into the endless waiting state.

The general solution for the elimination of the deadlock is to create the thread (StopThread) that calls the stop function separately and process the message in the main thread while waiting until the StopThread is ended using the message loop composed of MsgWaitForMultipleObjects().

2.3.3. Video Functions

Common in both Cap5 and Cod5 API sets. General setup function related to Video. The BOARD_INFO structure used by the function group supports following member variables.

Variable	Description
pResInfo	Supported resolutions by the board. Pointer to the RESOLUTION_INFO structure.

Video Format Functions

Uses enum VideoFormatValues. Currently only NTSC_M and PAL_B are supported. Configured in board level only. Configuration in channel level is not supported. Must be set before setting image size. Supports both Get/Set functions.

Image Size Functions

Setup image resolutions in both Raw Video and Compressed Video. In most cases, uses any resolution provided by channels. However, some special boards have limitation in the resolution, and support only a set of resolution. All channels in the same board must use resolutions from same resolution group. Detailed information about this limitation is available by pResInfo member variable of the RESOLUTION_INFO structure in any of CAP5_BOARD_INFO or COD5_BOARD_INFO structure. The BOARD_INFO can be retrieved by GetBoardInfo function of each API. Since the ImageSize depends on video format setting, must set the VideoFormat first. Both Get/Set functions are supported.

External Video Out Function

Select a channel to output through external video output port. Select desired channel per each board.

Video Status Function

Get video input status information of all channels in a board. While it is possible to passively get change information using the Event method, direct reading is also supported. Get the values in bitwise format. If the board supports, maximum 8 ULONG variables are supported (32 * 8 = 256 bits – channels).

2.3.4. Raw Video Functions

Only for the Cap5. No additional item in the BOARD_INFO structure.

Color Format Functions

The color format is detailed format of color data in the raw video. The color format need to be set in the Setup state of the driver. The driver support enum ColorFormatValues for the purpose. However, not all formats are supported by all drivers. Since each driver has limited capability of supporting color formats, please consult the specification before writing the APP. Supports both Get/Set functions

Description to major ColorFormat values

- YU12 is the easiest type in CODEC. Overlay support is reasonably suitable as well. Two pixels (vertical and horizontal) share one color information. Planar method. Uses 1.5byte per pixel. Both vertical and horizontal must use two pixel units together. Brightness and color information is separated..
- RGB method is the best method for GDI display output. RGB 24 uses 3 bytes per pixel. RGB555 (RGB15) and RGB565 (RGB16) use 2 bytes per pixel. The RGB24 has too big data compare to other formats while the RGB555 and RGB565 has two small bits which might cause gradation effect
- YUY2 is the best method for the Overlay type display and reasonably suitable for the CODEC

as well. Packeted method. Uses 1.5byte per pixel. Both vertical and horizontal must use two pixel units together. Brightness and color information is separated. The most common method.

- Y stores only black and white information. For special purpose of image processing, data size can be reduced to half of the YUY2 method by removing color information. Uses 1 byte per pixel.

Frame Rate Function

The FrameRate function has two modes. The first is letting the driver to automatically allocate maximum frame rates among channels (AutoEven). The Second is manually setting frame rate per each channel (UserFixed). The frame rate can be set per each channel.

AutoEven Mode: Equally distribute the maximum frame rate by the driver to all channels. For example, in the 16 channels 120fps model, share 7.5 fps per each channel. In this model, The AutoEven mode tries to maintain total 120fps. For example, if 8 of the 16 channels are unavailable due to no video signal or ChannelEnable(FALSE), the system shares the 120fps among the rest 8 channels to make each channel runs in 15 fps.
In the 16 channel 240fps model, if only 12 channels are captures, it is not possible for a channel to be captured by multiple VP. Therefore some channels run in 15fps while the others run in 30fps. This can be changed by change in number of captured channels. These are not requirements by normal DVR applications. It is best to manually assign the frame rate by the APP. The default mode is the AutoEven mode.

UserFixed Mode: Manually assign frame rate per each channel from the APP. Since the available frame rate is limited by the VideoFormat, ImageSize and hardware spec, only use the allowed frame rate in the APP. Even in the UserFixed mode, the assigned frame rate might not be available due to external signal, synchronization status, PCI range, delay in APP, PC speed, Mainboard chipset, and other external causes. Channels with 0 frame rate setting do not generate data. There fore, to actually generate data, all of following conditions must met.

- The driver is in the Run state (Limited only in the NCP series board)
- The channel is enabled
- Valid video signal
- The FrameRate is 1 or greater
- Free buffer

If any of the above conditions does not meet, actual capture does not occur.

2.3.5. DIO/WD Functions

Common for both Cap5 and Cod5 API sets. Other IO and control related functions. The BOARD_INFO structure related to the function group provides following member variables.

Variable	Description
uMaxDO	Maximum number of DO (Digital Out)
uMaxDI	Maximum number of DI (Digital In)

DIO

DIO is short of Digital In & Out. DI is short of Digital In. DO is short of Digital Out. Use DI to check on/off status of external devices (usually sensors). Use DO to turn on or off external devices (usually LED and etc.) To check number of installed DIO in the board, check MaxDI and MaxDO variables from the BOARD_INFO structure. Checking or setting the values for the DIO is done at board wide level in bitwise format. The DIO runs in either relay mode or powered mode depends on the board. For the accurate operation, change the mode to match the external device. Check the hardware manual for the detail

The DI can only get the status, while the DO can both set the status and get the recently set status. To turn on the external output (Relay), use the bit 0. DI has matching bit as 0 if the external device is turned on.

Turn on the relay: 0
Turn off the relay: 1
The sensor is turned on: 0
The sensor is turned off: 1

Checking status with the DI is available both in the Setup and Run states. Status change notification for the DI is only available in the Run state.

Both of Set/Get functions of the DO is available in both the Setup and Run status. To change status of a specific DO, use GetDO function or recently recorded DO value in the APP to get the current status. Then use bitwise operation to set the new value.

WatchDOG

Watchdog forces the system to hardware reset if the system malfunctions (The system does not reply for a given period of time). Some models have buzzer to notify user with beeps before the reset. Check the

hardware manual for whether or not supporting the buzzer.

When checking the reply, instead of making the driver check the system reply, make the APP to notify availability to the driver. If the driver does not receive the APP notification for a given time, the driver reset the system. The APP notification is called Triggering and the given period time is called Timeout.

The WatchDog operates only if enabled. The default timeout is 20 minutes. Setting the timeout value in disabled status does not cause the triggering. However, the watchdog is triggered when the system is turned enable. Setting new timeout value also stops triggering. The WatchDog functions are only available in the Run state. Several methods can be used for triggering in the APP. The most common method is using Window Timer while it is possible to make a separate thread to update the trigger.

I2C

These functions are used to manually control devices connected to I2C(IIC:Inter-IC) connected to the board. Not used in normal APP.

Following commands are supported.

CMN5_I2C_CMD_READ

CMN5_I2C_CMD_WRITE

CMN5_I2C_CMD_RANDOM_READ

CMN5_I2C_CMD_RANDOM_WRITE

CMN5_I2C_CMD_RANDOM_WRITE_WAIT

The CMN5_I2C_CMD_WRITE is a command that does not designate the address but writes only the data. Each board is supported differently. Therefore, it is recommended to use CMN5_I2C_CMD_RANDOM_WRITE.

2.3.6. Codec Functions

Only for the Cod5 API Set. Compression related functions. The BOARD_INFO structure related to the function group provides following member variables.

Variable	Descriptions
uCodecType	Type of generated stream
uVideoCodecType	Type of the Video CODEC
uAudioCodecType	Type of the Audio CODEC

The information transmitted to the COD5_BOARD_INFO is the default codec information. The codec that can be supported generally varies according to the board and more than two codecs can be supported. In case more than two codecs are supported, they can be set up using COD5_CPC_CODEC_TYPE of Cod5SetCodecProperty.

Codec Property

Provides Property Command to set the compression method when compressing video or video stream using codec. The Property Command can be set up only before initializing compression (setup-level), and it cannot be changed during compression. When changing it, stop the operation, then perform setup and start once again. The following table indicates how each command is used.

Property Command	Descriptions
COD5_CPC_STREAM_TYPE	The type of stream varies according to model and several streams can be created simultaneously. At least more than one value shall be selected to create a stream. For example, if the COD5_CST_VIDEO | COD5_CST_AUDIO is used, the Audio/Video stream is created. Different forms of codec are created according to the board and selected codec type. That is, although the video stream is created using the COD5_CST_VIDEO, if the selected codec is COD5_VCT_MP2, the MPEG2 video stream is created and if it is COD5_VCT_MP4, the MPEG video stream is created.
COD5_CPC_SKIP_FRAME	Designates the skip frame of the Compressed Video Stream. The actual frame rate becomes the total frames//(skipframe+1). If the frame rate is reduced, the actual bit-rate is reduced in proportion to the set bit-rate.
COD5_CPC_BITRATE	Sets up the bit-rate of the Mpeg Video Stream. Its unit becomes bps. The type of bit-rate setup includes constant bit-rate and variable bit-rate. The variable bit-rate is set up by designating average and maximum values, minimum and maximum values, and quantization value. Since each driver has a limited way of supporting, their contents must be checked first.
COD5_CPC_GOP_SIZE	Sets the GOP (Group of Pictures) size of the Mpeg video. Designates the size of all GOP and the number of B pictures.
COD5_CPC_AUDIO_ATTR	Sets the value for audio compression. It sets the sampling rate and the bit-rate.
COD5_CPC_CODEC_TYPE	Sets the codec to be used for compression.

Codec Adjust

Provides audio and video setup parameters that can be changed during compression. It provides a

command identical to the raw video and audio adjust command. However, the setup may be impossible according to the board characteristics due to HW-dependency.

2.3.7. Audio

The BOARD_INFO structure related to the function group provides folowing member variable.

Variable	Descriptions
uFrequencyType	Type of the Frequency group
uGainType	Type of the gain group

According to gain group type, the range of gain and default gain may be changed. The following table shows the gain range and default value as gain group type.

Gain Group	Gain Ranage
AUD5_AGT_GAIN_TYPE1	1-15, default 3
AUD5_AGT_GAIN_TYPE2	0-255 , default 128
AUD5_AGT_GAIN_TYPE3	1-4 range

2.3.8. Realtime

Only for the Ovr5 API set.

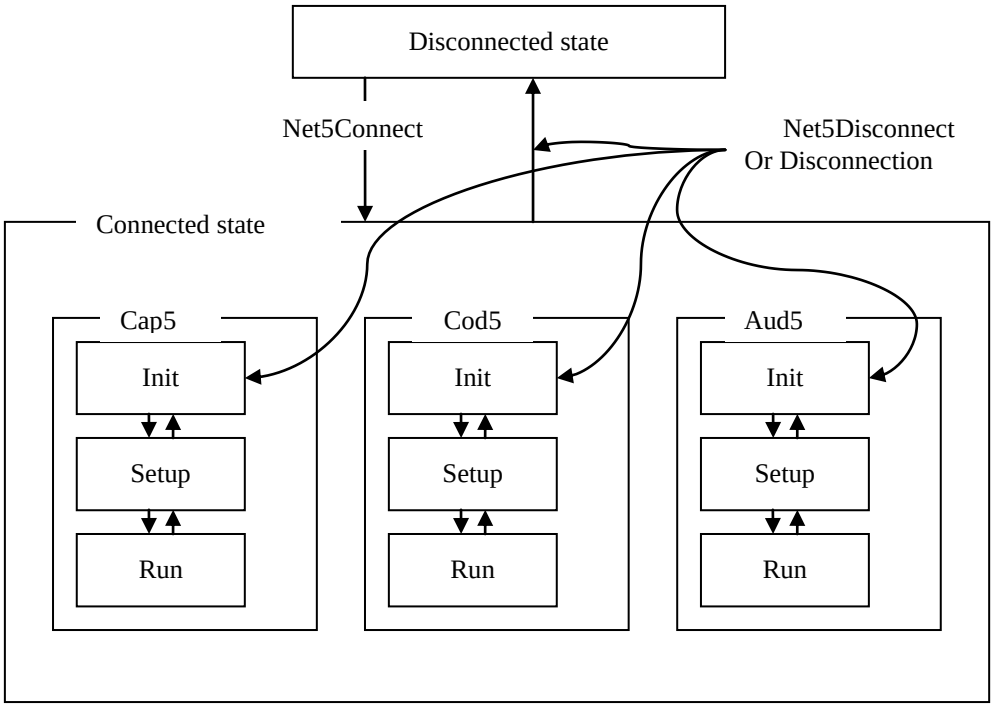
While most of the Ovr5 API functions are similar to the Video functions, slight difference due to characteristics of the Ovr separated this API set from the rest. Note that there are some differences regardless of similarity to the Video API.

2.3.9. Network

Only for the Net5 API set

The network device can also use API including Cod5 and Cap5. However, they can be used after connection is made using the Net5 API.

The following figure presents the diagram for the correlation between the Connection State/Process Control State and each connection. As can be seen from the figure, the connection state affects the process control state of the remaining API.


Disconnected state
Net5Connect
Net5Disconnect
Or Disconnection
Connected state
Cap5
Init
Setup
Run
Cod5
Init
Setup
Run
Aud5
Init
Setup
Run

3. Using Version 5 API

Chapter 3 describes the actual API usage.

3.1. Preparing functions calls

Several preparation procedures are needed to call the actual API function. They are divided into two methods according to how the API is called. The first method is to call the exported function of dll and the other one is to call the method by creating the instance of the COM Interface. Since the concrete method for the above differs according to the language, each method is considered for each language. However, the method to use the export function cannot be used for the network device due to the API structure.

Both the above two methods also require a process to obtain an appropriate dll name according to the set device.

Using Exported Function(Dynamic link)

In order to use the function exported to DLL, the pointer of each function exported to the corresponding dll can be imported using the LoadLibrary()/GetProcAddress()/FreeLibrary() functions of Win32.

To use the dynamic link, retrieve each API function as a function pointer. Three steps are required to do this in the source code: defining function pointer variable, declaring function pointer variable, and invoking function pointer address. Following are steps required to use the function pointer.

- Get required DLL filename and path
- Use LoadLibrary() to load the DLL and get the handle
- Get address of each function.
- Use the function pointer to use the function
- Unload the DLL

Using Com interface methods.

Although it supports the com interface, it cannot be created using standard com library such as cocreateinstance due to the intricacy of the dll configuration. The Cmn5createinstance function should be used. The method to use the Cmn5createinstance is identical to the method to use the export function mentioned above.

Local Device and Net Devices

Although the local device performs createinstance several times, it will return the same interface pointer,

which is a meaningless operation. For the net devices, since each created instance corresponds to each connection by 1:1, the instance shall be created for each net device.

Using COM interface has several advantages. The first is that you do not have to load dozens of exported API function using GetProcAddress. You can get them using only QueryInterface.
The second is that you can find out API set a DLL provides. For example, ECPSV.DLL provides Cap5, Aud5, Ovr5. MPJVS.DLL provides Cod5. You have difficulty, if you use exported API method. But, if you use COM interface method, it is easy because you just QueryInterface for a specified DLL.

3.1.1. In C++

Files for C++

The following files are included according to the API set to be used.

API	File name
Common	Cmn5BoardLibEx.h(Must be included)
Cap5	Cap5BoardLibEx.h
Cod5	Cod5BoardLibEx.h
Aud5	Aud5BoardLibEx.h
Ovr5	Ovr5BoardLibEx.h

Using Exported Function in C++

Followings are the example of the Dynamic Link in c++.

Following is declaring the function pointer variable. Include it to all *.cpp files using it.

```
Extern BOOL CMN5_API (*_Cod5Setup)(void);
extern BOOL CMN5_API (*_Cod5Endup)(void);
extern BOOL CMN5_API (*_Cod5Run) (void);
extern BOOL CMN5_API (*_Cod5Stop)(void);
```

Following is defining the function pointer variable and retrieving function pointer value.

```
BOOL CMN5_API (*_Cod5Setup)(void);
BOOL CMN5_API (*_Cod5Endup) (void);
BOOL CMN5_API (*_Cod5Run) (void);
BOOL CMN5_API (*_Cod5Stop) (void);
HMODULE g_hDll;

LoadDriver()
{
    g_hDll = LoadLibrary("mp204.dll");
    _Cod5Setup=(BOOL CMN5_API (*)(void))GerProc(g_hDll, "Cod5Setup");
    _Cod5Endup=(BOOL CMN5_API (*)(void))GerProc(g_hDll, "Cod5Endup");
    _Cod5Run=  (BOOL CMN5_API (*)(void))GerProc(g_hDll, "Cod5Run");
    _Cod5Stop= (BOOL CMN5_API (*)(void))GerProc(g_hDll, "Cod5Stop");
}

UnlooadDriver()
{
    FreeLibrary(hDll)
}
```

Following is an actual usage code.

```
    _Cod5Setup();
    _Cod5Endup();
    _Cod5Run();
    _Cod5Stop();
```

Note that function call using a function pointer can be done in following methods, but the first method is more convenient.

```
_Cod5Setup();
(*_Code5Setup)();
```

Using COM interface methods in C++

```
static BOOL Uda5CreateInstance(HMODULE hLib,REFIID riid,void ** ppInterface)
{
    if(hLib){
        BOOL rs;
        Iunknown* pUnknown;
        BOOL (FAR WINAPI*_CreateInstance)(Iunknown ** ppInterface);
```

```
        FARPROC test_proc=GetProcAddress(hLib,"Cmn5CreateInstance");
        if(test_proc){
            *(FARPROC*)&_CreateInstance=test_proc;
            rs=(*_CreateInstance)(&pUnknown);
            if(rs){
                HRESULT hr;
                hr=pUnknown->QueryInterface(riid,ppInterface);
                pUnknown->Release();
                if(SUCCEEDED(hr))
                    return TRUE;
            }
        }
    }
    return FALSE;
}

HRESULT CMPJVCardManager::Load(...)

...

    Icod5*  pCod5Api;
    Icap5*  pCap5Api;
    Iaud5*  pAud5Api;
    hLib=LoadLibrary("MPJVS.DLL");

    if (!Uda5CreateInstance(hLib, IID_Icod5, (void**)&pCod5Api)) {
        return E_FAIL;
    }
    pCod5Api->QueryInterface(IID_Icap5, (void**)&pCap5Api);
...

}
```

The actual API is used in the following manner.

```
Void SomeFunction()
{
    // COM interface was acquired from other function.

    M_pCap5Api->Cap5Setup();
    m_pCap5Api->Cap5Run();

    m_pCod5Api->Cod5Setup();
    m_pCod5Api->Cod5Run();

    //
}
```

3.2. Using Driver's API

The chapter 3 describes actual standard API using procedure.

This paragraph describes API usage for actual API usage sequence. Examples are provided for general usage. API not described in the paragraph can be used in the same way. Some scenarios for the program configuration will be described and then the event method will be detailed more concretely.

3.2.1. Program Configuration Scenario

This chapter describes three methods that are most frequently used among the methods to configure the program that uses API.

Program Implementation by Callback Method

When using the callback method, the program configuration scenario is as follows: When using callback method, most programs are configured in the following manner.

1. Load the driver dll.
2. Get the driver information.
3. Activate the driver.
4. Perform driver setup.
5. Register the callback function.
6. Set up various properties/adjusts.
7. Run the driver.
8. Now, the driver calls the StartCallback, DataCallback and EndCallback registered by the APP. The DataCallback is called each time the data is generated.
9. Create the thread that stops the driver.
10. Wait until the stopthread ends while processing the message.
11. End the driver up to exit or repeat the above procedure from the step number 6.

Program Implementation by Event Method 1

Event method is used. Processing is performed in one independent thread. Generally, the network device creates a thread for each connection and performs processing as follows:

1. Load the driver dll.
2. Make connection in the case of network device.
3. Get the driver information.

4. Activate the driver.
5. Perform driver setup.
6. Set up various properties/adjusts.
7. Run the driver.
8. Wait for event.
9. If an event occurs, obtain, process and release the data.
10. Change the adjust. Repeat the above procedure from step number 8 until it must be stopped.
11. Stop the driver.
12. End the driver or repeat the above procedure from step number 6.
13. Disconnect in the case of the network device.

Program Implementation by Event Method 2

In this method, the initial endup is performed in the main thread and separate thread that processes only the data that is created and used.

In the main thread:

1. Load the driver dll.
2. Get the driver information.
3. Activate the driver.
4. Perform driver setup.
5. Set up various properties/adjusts.
6. Create dataloop thread.
7. Run the driver.
8. Stop the driver.
9. End the dataloop thread.
10. End the driver up to exit or repeat the above procedure from step number 5.

In the data loop thread:

1. Wait for event.
2. If an event occurs, obtain, process and release the data.
3. Repeat the above procedure from step number 1 until it must be stopped.

3.2.2. Getting Started

General programming structure are described in the next. See next paragraphs for details. The DataLoop

type has different usage details and may not follow the example.

```
Void main()
{
    //OnCreate,OnInitDialog
    GetDriverInformation();//refer to 3.3.2. Driver Informations

    //On User pushed the start button
    OpenDriver();//Create DataLoopThread

    //
    //finishes when the user push the stop button
    UILoop();//or Message Loop

    //on user pushed the stop button
//OnDestroyWindow
    CloseDriver();//Terminate DataLoopThread
}

//refer to  3.3.5. ~3.3.8 Data Loops
DataLoopThread()
{
    DataLoop;
};

void OpenDriver()
{
    InitializeDriver();//refer to 3.3.3.     Driver Initialization
    StartDriver();//refer to 3.3.4.   Start Driver
}

void CloseDriver()
{
    StopDriver();//refer to 3.3.9.    Stop Driver
    UninitializeDriver();//refer to 3.3.10.  Driver Uninitialization
}
```

3.2.3. Driver Information

Use the GetSystemInfo function to collect driver initialization information. Get number of boards and driver information of the API. Then collect information for each board. The step is required for each API. Call all functions in each API to use. Below is an example for collecting information of the Cap5 API and Cod5 API.

```
BOOL GetDriverInformation()
{
    Cap5GetSystemInfo(&g_CapSystemInfo)
```

```
    Cod5GetSystemInfo(&g_CodSystemInfo)

    ULONG I;
    CMN5_BOARD_INFO_DESC bid;
    bid.uInfoVersion   = CMN5_BOARD_INFO_VERSION;
    bid.uInfoSize  = sizeof(CAP5_BOARD_INFO);
    for(i=0; i<g_nCapBoards; i++){
        Cap5GetBoardInfo(I, &bid, &g_pCapBoardInfo[i]);
    }
    bid.uInfoSize  = sizeof(COD5_BOARD_INFO);
    for(i=0; i<g_nCodBoards; i++){
        Cod5GetBoardInfo(I, &bid, &g_pCodBoardInfo[i]);
    }

    return TRUE;
}
```

3.2.4. Driver Initialization

The Driver Initialization means initializing each board with basic driver information and separately acquired Activation code, then calling the Setup function. Each API requires the driver initialization. Below is an example of initializing the Cap5 API and Cod5 API. The GetActCodefromModelID function returns Activation Code from ModelID provided by the GetBoardInfo function.

```
BOOL InitializeDriver()
{
    BOOL rs;
    UCHAR activeCode[16];
    for(int i=0;i<(int)g_CapSystemInfo.uNumOfBoard;i++) {
        GetActCodefromModelID(g_pCapBoardInfo[i].uModelID, activeCode);
        rs = Cap5Activate(I, g_ActivationCode);
        if(!rs) {
            return FALSE;
        }

        GetActCodefromModelID(g_pCodBoardInfo[i].uModelID, activeCode);
        rs = Cod5Activate(I, g_ActivationCode);
        if(!rs) {
            return FALSE;
        }
    }

    if(!Cap5Setup()){
        return FALSE;
    }

    if(!Cod5Setup()){
```

```
        return FALSE;
    }

    return TRUE;
}
```

The GetActCodefromModelID function uses an internal array of Activation Code to ModelID to return matched ActiveCode from parameter ModelID. ModelID and ActiveCode in the next example is not actual values. The actual values are provided separately from the company.

```
Struct CMN_MODELID_ACTCODE {
    DWORD ModelID;
    BYTE ActCode[16];
};

CMN_MODELID_ACTCODE g_IdnCode[] = {
    {0x08A1, { 0x21, 0x49, 0x34, 0xBC, 0x3C, 0x8D, 0x90, 0xD3,
               0xF5, 0x73, 0x22, 0xA0, 0xA1, 0x3D, 0x2C, 0x58}},
    {0x08A2, { 0x34, 0xBC, 0x3C, 0x8D, 0x03, 0xF5, 0x73, 0x1F,
               0x02, 0xC0, 0xA1, 0x3D, 0x02, 0x7C, 0x9E, 0x02}},
    {0x08A3, { 0x58, 0x03, 0xA8, 0x03, 0xF5, 0x73, 0x22, 0xA0,
               0x49, 0xC3, 0x59, 0xC0, 0xA1, 0x03, 0x03, 0x03}},
};

BOOL GetActCodefromModelID(DWORD ModelID, UCHAR * ActCode)
{
    for(int i=0;i<sizeof(g_IdnCode)/sizeof(g_IdnCode);i++) {
        if(g_ModelIDnActCode[i].ModelID== hwid) {
            CopyMemory(ActCode, g_IdnCode[i].ActCode,16);
            return TRUE;
        }
    }
    return FALSE;
}
```

3.2.5. Start Driver

This paragraph describes how to actually running the driver software. The next code is an actual example. The ApplySetting function assigns properties and adjusts for each API. Details are explained later. The RunCaptureThread function loads generated data and create /starts processing DataThread. Internal structure is described in the next paragraph. Create the DataThread and call Run function of each API to actually start the driver. Prepare data processing before running the driver to prevent data loss. Otherwise, generated data before starting the DataThread may cause buffer overflow of the driver. The StartDriver function also starts the WatchDog. In the example, since the Window Timer is used to drigger the

WatchDog, call the SetTimer function to create the WindowTimer. The APP_WATCHDOG_TIME_PERIOD must be smaller than the WatchDog Timeout value.

```
BOOL StartDriver()
{
    ApplySetting();

    RunDataLoopThread();

    Cod5Run();
    Cap5Run();

    Cap5SetWatchDog(0, WC_ENABLE,0,0);
    Cap5SetWatchdog(0, WC_SET_TIMEOUT_VALUE, 30, 0);
    Cap5SetWatchdog(0, WC_SET_BUZZER_TIMEOUT_VALUE, 1, 0);
    SetTimer(WATCH_TIMER_ID,10*1000,NULL);   // each call 10s

    return TRUE;
}
```

The ApplySetting function includes routine for setting all property required in the driver setup state. Generally, configuration dialog in the APP is used to collect user inputs. The function also calls Adjust functions to set initial Adjust for generated data from the Run. Each setting can be applied to board or channel level. Use appropriate loops algorithm. The next example adjusts VideoFormat, ColorFormat, ImageSize, and Video in the Cap5, while setting ImageSize in the Cod5. In the actual APP, all properties/adjusts by the API must be expressly set. Otherwise, default value in the drive may cause unexpected data.

```
BOOL ApplySetting()
{
    for(DWORD bd=0;bd<g_nCapBoards;bd++) {
        Cap5SetVideoFormat(bd,videoformat);
        Cap5SetColorFormat(bd,CAP_COLOR_FORMAT_YUY2);
        for(DWORD i=0;i<CapNumOfChannel(bd);i++) {
            Cap5SetImageSize(bd,I,chprop.imagesize);

            Cap5SetAdjust(bd, I, VAC_BRIGHTNESS,brightness,0,0,0);
            Cap5SetAdjust(bd, I, VAC_CONTRAST,contrast,0,0,0);
            Cap5SetAdjust(bd, I, VAC_SATURATION_U,saturationu,0,0,0);
            Cap5SetAdjust(bd, I, VAC_SATURATION_V,saturationv,0,0,0);
            Cap5SetAdjust(bd, I, VAC_HUE,hue,0,0,0);
        }
    }

    for(bd=0;bd<g_nCodBoards;bd++) {
```

```
        for(DWORD i=0;i<CapNumOfChannel(bd);i++) {
            ULONG codImgSize= (vidfmt == CMN_VIDEO_FORMAT_NTSC_M)?
                MAKEIMGSIZE(640, 480):MAKEIMGSIZE(720, 576);
            Cod5SetImageSize(bd,I,codImgSize);
        }
    }

    return TRUE;
}
```

The RunDataLoopThread generates the DataThread. The function generates one thread in the Event method. Below is now to generate the thread in the Event method.

```
BOOL RunDataLoopThread ()
{
    unsigned id;
    m_hDataLoopThread =  _beginthreadex(0, 0, DataLoopThread, 0, 0, &id);
    return TRUE;
}
```

The next is how to generate the thread in the Query method using Cap5 and Cod5 API sets.

```
BOOL RunDataLoopThread ()
{
    unsigned id;
    m_hCapDataLoopThread =  _beginthreadex(0, 0,
                              CapDataLoopThread, 0, 0, &id);
    m_hCodDataLoopThread =  _beginthreadex(0, 0,
                              CodDataLoopThread, 0, 0, &id);
    return TRUE;
}
```

3.2.6. Data Loop

This chapter shows example of Data Loop using the Event method.

In the Event Data Loop, one thread handles all data types. Since only one thread is used, race condition or synchronization problem found in the Query method are not found. The next is general Event type DataLoopThread. FetchAndProcessRawVideoData and FetchAndProcessCompressedData are explained in the next paragraph. HstopEvent comes in the top to test thread-ending condition. Get all event handles for the DataType to use and store them in the array. Use Win32 WiatForMultipleObject function to check DataEvent or Ending Event. If the Ending event is set, end the thread. If the DataEvent is set, process appropriate processing depends on the DataType. Simple status changes such as DT_SENSOR and

DT_VSTATS do not require calling the ReleaseData function.

```
DWORD WINAPI DataLoopThread ()
{
    HANDLE events[5];
    BOOL retVal;

    events[0] = hStopEvent;
    Cap5GetEventHandle(DT_VSTATUS,&events[1]);
    Cap5GetEventHandle(DT_SENSOR, &events[2]);
    Cod5GetEventHandle(DT_COD,    &events[3]);       //cod
    Cap5GetEventHandle(DT_VIDEO,  &events[4]);

    while(1){
        DWORD obj = WaitForMultipleObjects(5,events, FALSE, INFINITE);

        if(obj==WAIT_OBJECT_0) break;//terminate this thread.

        Switch(obj){
        case WAIT_OBJECT_0+4:
            FetchAndProcessRawVideoData();
            break;
        }
        case WAIT_OBJECT_0+3{
            FetchAndProcessCompressedData();
            break;
        }
        case WAIT_OBJECT_0+2{
            CMN5_SENSOR_STATUS_INFO info={0,};
            if(Cap5GetEventData(d_type, (void*)&info)) {
                OnSensor(&info);
            }
            break;
        }
        case WAIT_OBJECT_0+1{
            CMN5_VIDEO_STATUS_INFO info={0,};
            if(Cap5GetEventData(d_type, (void*)&info)) {
                OnVideoPresent(&info);
            }
            break;
        }
        }//switch
    }// while
    return 0;
}
```

In the WaitForMultipleObject function, while waiting for multiple events simultaneously, using fixed sequence of events can cause the following problem. If a preceding event is more frequent than other

events in a fixed sequence, data generated by other events with lower priority might be temporarily lost when the data buffer is full. To avoid this while keeping the fixed sequence, move the frequent events to the end of the sequence to prevent the problem.
However, for the better resource distribution, you can move any event to end of the sequence whenever the event generates data.

For MediaData, the driver has internal cue to store information to the generated data. Event for the type means there is one or more data in the cue. If the Event is set by the Wait function, the event is reset. If there is one or more data in the cue, the next data becomes unavailable. Therefore a variable uHasNextData is provided to notify that there are more data in the cue. Run a loop to completely empty the cue while the uHasNextData is set. The APP uses the function to display, compress, or process raw video data. Release buffer for the processed data with the Cap5ReleaseData function.

```
Void FetchAndProcessRawVideoData()
{
    do {
        CAP5_DATA_INFO info={0,};
        if(retVal = Cap5GetEventData(d_type, &info)) {
            if(info.pDataBuffer) {
                OnCapture((CMN5_DATA_INFO_HEADER*)&info);
            }
            Cap5ReleaseData((void*)&info);
        }
    } while(retVal && info.uHasNextData);
}
```

For RawVideoData, data is generated per frame. Even if FramDrop occurs, the driver keeps generating data. Therefore no additional action is required. However, for CODEC Data, if an error occurs, the driver stops data generation from the channel. The APP needs to perform appropriate processing and enable the driver with the Cod5VideoEnable function. The next is example of Cod5 data receiving.

```
Void FetchAndProcessCompressedData()
{
    do {
        COD5_DATA_INFO info={0,};
        if(retVal = Cod5GetEventData(d_type, &info)) {
            if(info.uErrCode!= EC_NO_ERROR) {
                Cod5VideoEnable(info.uBoardNum, info.uChannelNum, TRUE);
            } else {
                if(info.pDataBuffer) {
                    OnCapture((CMN5_DATA_INFO_HEADER*)&info);
                }
                Cod5ReleaseData((void*)&info);
```

```
            }
        }
    } while(retVal && info.uHasNextData);
}
```

3.2.7. Data Process 1– Media Data

The Media Data pointer is to data position and information about the data. The Media Data also includes the board number and the channel number. Beginning of the Media Data structure is equal to the CMN5_DATA_INFO_HEADER structure. Check uDataType field and cast the data to each API structure.

```
BOOL OnCapture(CMN5_DATA_INFO_HEADER* pdata)
{
    int bdid = pInfoData->uBoardNum;
    int bdch = pInfoData->uChannelNum;

    if(pdata->uDataType==DT_VIDEO){
        CAP5_DATA_INFO* pInfoCod = (CAP5_DATA_INFO*)pdata;
        PBYTE Image = (pInfoData->pDataBuffer);
        if(Image) {
            DisplayImage(bdid,bdch,Image);
        }
    }else if(pdata->uDataType==DT_COD){
        COD5_DATA_INFO* pInfoCod = (COD5_DATA_INFO*)pdata;
        PBYTE Image = (pInfoData->pDataBuffer);
        if(Image) {
            g_recstore.StoreImage(bdid,bdch,Image, pInfoCod->uDataSize);
        }
   }else{
        //process other data types...
    }

    return TRUE;
}
```

3.2.8. Data Process 2- Status

Process the Status change in the board level. While the status contains the current status and changed channel information, only process the changed channels. The next is example of processing the Video Status change. Since the API 5 supports up to 256 channels, all bits in the 8 ULONG variables must be checked.

```
BOOL OnVideoPresent(CMN5_VIDEO_STATUS_INFO* pStatus)
{
    ULONG bd = pStatus->uBoardNum;
    ULONG nChannel = CapNumOfChannel(bd);
```

```
    for(ULONG i=0;i<nChannel;i++){
        int index= i/32; //32 means ULONG'S bit count
        int bit=i%32;

        if(pStatus->VideoStatusMask[index]&(1<<bit)){
            BOOL  isPresent= (pStatus->VideoStatus[index]&(1<<bit))?1:0;

            ProcessVideoStatusChange(bd,I,isPresent);
        }
    }
    return TRUE;
}
```

Following is an example of processing Sensor (DI). The method is similar to the Video Status processing.

```
BOOL OnSensor(CMN5_SENSOR_STATUS_INFO* pStatus)
{

    ULONG bd = pStatus->uBoardNum;
    ULONG nSensor = g_CapBoardInfo[bd].uMaxDI;

    for(ULONG i=0;i<nSensor;i++){
        int index = i/32; //32 means ULONG'S bit count
        int bit = i%32;

        if(pStatus->SensorStatusMask[index]&(1<<bit)){
            BOOL  bSensorSet = (pStatus->SensorStatus[index]&(1<<bit))?1:0;

            ProcessSensorStatusChange(bd, I, bSensorSet);
        }
    }
    return TRUE;
}
```

3.2.9. Stop Driver

Corresponds to the Start Driver. Includes stop condition in the DataLoopThread.

```
BOOL CmainControlDlg::StopCapture()
{
    StopDataLoopThread();

    KillTimer(WATCH_TIMER_ID);
    Cap5SetWatchdog(0, WC_DISABLE, 0, 0);

    Cod5Stop();
    Cap5Stop();
```

```
}
```

The StopCaptureThread stops the DataThread. In the Event method, there is only one thread. To stop the thread, set end event and wait for the thread to stop.

```
Void StopDataLoopThread()
{
    SetEvent(m_hStopCaptureEvent);
    WaitForSingleObject(m_hThreadLoopProc, INFINITE);
    CloseHandle(m_hCaptureThreadProc);
}
```

In the Query method, instead of using the end event, set the g_bCapture variable to 0 and wait for all DataLoopThread to stop.

```
Void StopCaptureThread()
{
    g_bCapture = 0;
    WaitForSingleObject(m_hCapDataLoopThread, INFINITE);
    WaitForSingleObject(m_hCodDataLoopThread, INFINITE);
    CloseHandle(m_hCapDataLoopThread);
    CloseHandle(m_hCodDataLoopThread);
}
```

3.2.10. Driver Uninitialization

The counterpart of Driver Initialization. Only calls Endup functions of each API.

```
BOOL UninitializeDriver()
{
    Cap5Endup();
    Cod5Endup();
    return TRUE;
}
```

Revision History

Date	Revision	Description
2004-02-24	A	Preliminary
2005-06-08	B	Second Revision
2006-02-24	C	Third Revision
2006-08-21	D	Do not support Delphi anymore.
2008-04-11	E	Caution about GetEventData and ReleaseData added
2008-07-21	F	Model examples in API set outline corrected
2009-04-27	G	Correct the errata
2009-05-15	H	Added the Microsoft Windows Vista to the supported OS

It is apt to change the contents.