SDK-10000

C Library Edition

V1.2.45

Programming Guide

Table of Contents

1

Overview

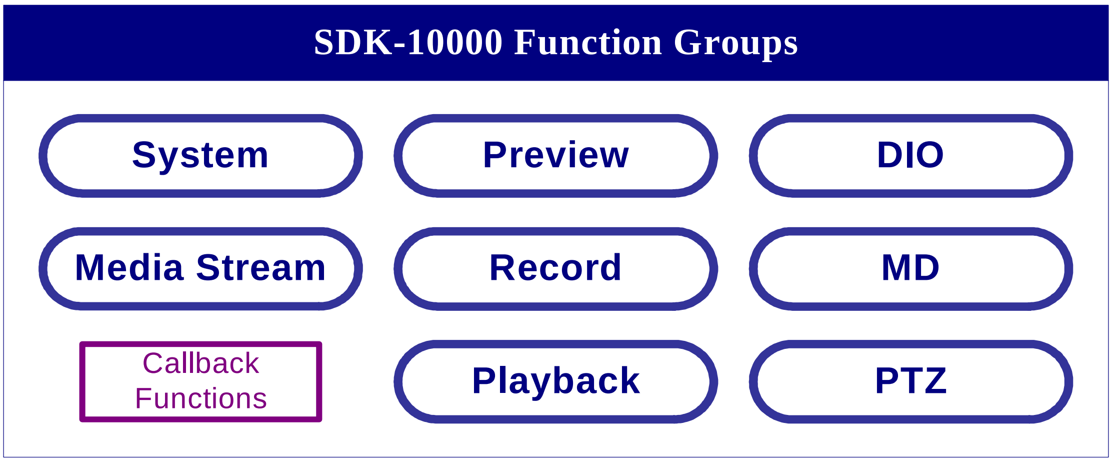
Introduction

This material covers SDK architecture, data structure and procedures to illustrate the mechanisms to integrate the IP Surveillance devices. The content of this material is designed to lead the programmers go through the flow of the SDK and design their own application with supplied functions; they are organized in topics so that programmers may find the topics they want directly.

Please refer to Programming Guide for detailed API references.

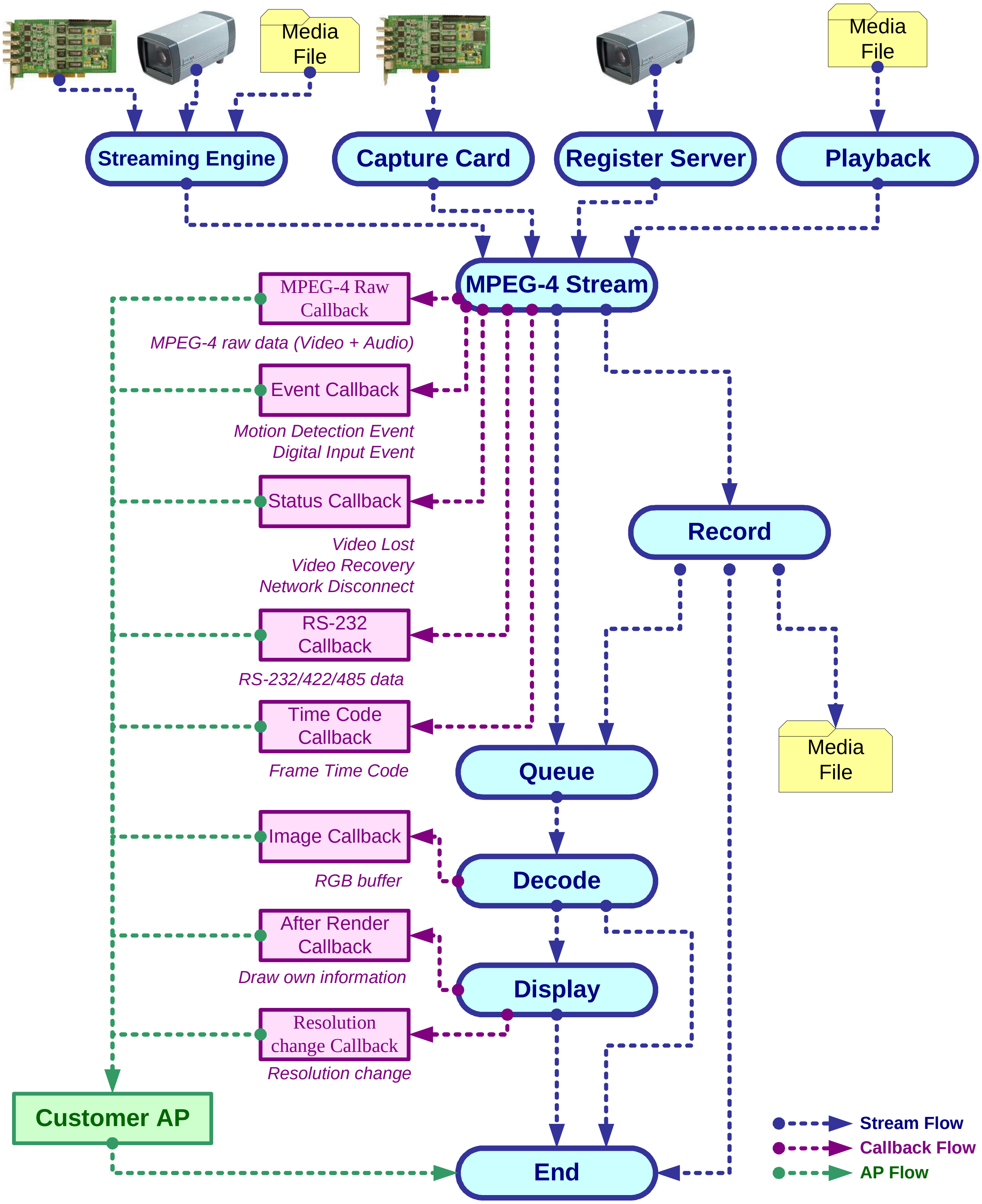
SDK Function Groups

The whole SDK can be divided into following function groups.



Architecture

SDK architecture and data flow is described as follow:



Application Type

Based on the architecture and data flow, users may develop following application type:

1. **Full-featured Surveillance system**: preview, record, playback, DIO event, MD event and PTZ functions
2. **Background recording**: record without preview. The stream can be configured as unicast or multicast mode
3. **Connection with event handling only**: connection only, wait for digital input or motion detection event; when the event triggered, then starts streaming and record the event
4. **Background recording with RGB buffer**: record without preview, receives RGB buffer to run user-defined motion detection algorithm at the same time
5. **Process MPEG-4/Motion JPEG/H.264 video stream**: advanced users may acquire video stream and process by themselves. Related video, audio and audio+video callback functions are provided
6. **User-defined information on screen**: user may use after render callback function to draw user-defined information on preview window, including OSD text, draw video intelligence information

Topics

Streaming Client Library is developed for MPEG-4/Motion JPEG/H.264 Video Network Streaming Application.

It contains following abilities:

- Registration with Unicast / Multicast
- Preview / Record / Playback
- DIO Event Handling
- Motion Detection Event Handling
- PTZ Integration
- Status Callback
- IP Quad Integration
- Advanced Topics
    - Gets Video data via Video callback function
    - Gets RGB via image callback function
    - Video Time code format
    - Decode I Frame Only
    - Save Video raw data into AVI format
    - Gets RGB via image callback function

What’s New?

Following lists the new contents in this release:

v1.2

- Multiple-files playback.
- Fast file shuffle playback with index
- Auto frame rate by CPU thrash hole
- Inverse rendering image.
- New PTZ functions and absolute PTZ functions.
- Support Motion-JPEG
- Support H.264
- Support Mega-pixel mpeg4
- Jitter less function
- New Callback functions
- New Codec Type: IPP

v1.2sp1

- New chapter about H.264/RTP
- Delete all chapters about URL command. (Attach “URL Command Specification” to SDK folder instead)
- Support Dual-Stream connection.(Need Dual-Stream Devices)

SDK-10000 Key Features

Category	Function	Files	SDK-10000		
			v1.1	v1.2	v1.2 SP1
Connection	TCP v1.0	ATCP10.dll	v	v	v
	TCP v2.0	ATCP20.dll	v	v	v
	Multicast v1.0	AMCST10.dll	v	v	v
	Multicast v2.0	AMCST20.dll	v	v	v
	Streaming Engine	ASE.dll	v	v	v
	RTP over UDP	ARTP.dll		v	v
	RTP over MCST	ARTP.dll		v	v
Resolution	Megapixel	KMPEG4.dll		v	v
H.264 Decoder	Intel IPP	IPPCodec.dll		v	v
MPEG-4 Decoder	XviD	XVIDCodec.dll	v	v	v
	FFMPEG	FFMCodec.dll	v	v	v
	Intel IPP	IPPCodec.dll		v	v
MJPEG Decoder	MJPEG Decoder	MJPEGCodec.dll		v	v
Render	GDI	DGDI.dll	v	v	v
	DirectX	DxDraw.dll		v	v
Kernel	DIO	KMPEG4.dll	v	v	v
	Motino Detection	KMPEG4.dll	v	v	v
Live View	Decode I-Frame	KMPEG4.dll	v	v	v
	Auto Drop Frame	KMPEG4.dll		v	v
	Flip	KMPEG4.dll		v	v
	Mirror	KMPEG4.dll		v	v
	Privacy Mask	KMPEG4.dll		v	v
Record	RAW format	FRAW.dll	v	v	v
	AVI format	FAVI.dll	v	v	v
	RAW + IDX format	FRAW2.dll		v	v
	Record H.264	FRAW.dll , FRAW2.dll			v
	Record MJPEG	FRAW.dll , FRAW2.dll			v
Playback	RAW format	ARAW.dll	v	v	v
	Play multiple file	AMRAW.dll		v	v
	Play H.264	ARAW.dll,AMRAW.dll			v
	Play MJPEG	ARAW.dll,AMRAW.dll			v
PTZ	PTZ Operation	PTZParser.dll	v	v	v
	Absolute Position	PTZParser.dll		v	v

Details:

(v1.2.37) Add function of KSetVideoTransferConfig, KSetMotionInfoEx, KGetMotionInfoEx.

(v1.2.37) The function of KSetTextOut is work well.

(v1.2.36)

a. Add TCPVideoStreamID to specify video track, value 0 to 255 for 1 to 256 video track.
b. Add RTPVideoTrackNumber (set it to 0, ARTP will use 1st video track, 1 to 255 is for specify video track).
c. Add RTPAudioTrackNumber (set it to 0, ARTP will use 1st audio track, 1 to 255 is for specify audio track).

(v1.2.35) ARAW has been supported time zone.

(v1.2.35) Removed VideoTrackIDOnRTP and AudioTrackIDOnRTP, and change ChannelNumber size to integer.

(v1.2.34) Removed StreamID and using ChannelNumber instead.

(v1.2.33) Handle H.264, MJpeg resolution change.

(v1.2.33) Enable Mjpeg Decode I Only (1 Frame per second).

(v1.2.31) Add KSetSmoothFastPlayback for smooth fast forward playback.

(v1.2.30) Add KGetDIOStatusByHTTPEx to request DIO status from multi-channel Devices

(v1.2.28) Support full time zones.

(v1.2.27) Support H.264, Mpeg4, Mjpeg on preview, record and playback.

(v1.2.27) Support Dual-Stream connection.(Need Dual-Stream Devices).

(v1.2.27) Replace time code by local time (default setting), to use KReplaceTimeCodeByLocalTime to enable / disable the function.

(v1.2.27) Add KDropNextPFrameTillIFrame for drop decoding of P-frames in a GOP.

(v1.2.19) Supporting 16ch preview (D1 @ 30 FPS 1.5MB bit rate)

(v1.2.18) Supporting preview, record and playback for H264 and MJpeg.

(v1.2.17) Add contact type CONTACT_TYPE_MULTIPLE_PLAYBACK for multiple playback and remote multiple playback.

(v1.2.16) Add KSetAutoDropFrameByCPUPerformance to enable auto drop frame mode. It enable a CPU thrash hole ensure dynamic frame rate.

(v1.2.16) Add KSetTimeCodeCallbackEx to call back time code in millisecond (v1.2.16) Add KSetFirstB2Callback to call back first B2 packet

(v1.2.16) Rename KReverseImageLeftToRight to KMirrorImage.

(v1.2.16) Rename KReverseImageUpToDown to KFlipImage.

(v1.2.15) Add Multiple Files Playback functions

API :

- KSetMultipleMediaConfig
- KAddMultipleMedia
- KRemoveMultipleMedia
- KClearAllMultipleMedia
- KGetNthBeginTimeFromMultipleMedia
- KGetNthEndTimeFromMultipleMedia
- KGetTotalIFramesOfMultipleMedia
- KGetCurrentReadingFileIDFromMultipleMedia
- KGetCurrentReadingAbsTimeFromMultipleMedia

(v1.2.13) Add KPTZGetRequestAbsPTZCommand for get a PTZ command to request PTZ absolute position.

API :

- KPTZGetRequestAbsPTZCommand

(v1.2.13) Add KSetFirstB2Callabck for get the first B2 data.

API :

- KSetFirstB2Callabck

(v1.2.12) Modified the callback data of KSetDICallbackEx from int *32 to an array

(v1.2.12) Add absolute position of PTZ functions.
Two new samples demonstrate the new functions.

(v1.2.12) Add an error code value by 32 for streaming fail

(v1.2.10) Add KEnablePricavyMask for setup 3 region of privacy mask on preview

(v1.2.08) Add KSetDICallbackEx to notify DI on / off

API :

- KSetDICallbackEx

(v1.2.06) Support mega-pixel mpeg4 video.

(v1.2.06) Support mega-pixel motion jpeg video (preview only)

(v1.2.06) Add decoder mode in SDK10000, now you can use SDK10000 to decode the Mpeg4 video from IP camera.

API :

- KStartDecodeMode
- KDecodeFrame
- KStopDecodeMode.

(v1.2.06) Add Digital PTZ functions.
Digital PTZ functions.

API :

- KDigitalPTZEnable
- KDigitalPTZTo

(v1.2.06) Add Reverse image left to right function

Inverse the image while playing video stream.

API :

- KReverseImageLeftToRight
- KReverseImageUpToDown

(v1.2.06) Add Jitter less adjust function
Enable a buffer to keep down jitter of video stream.

API :

- KEnableJitterLessMode
- (v1.2.06) Increase connecting speed.

▪

Compiling and Linking

This section describes the compiling and linking options.

Include Files ${SDK DIR}\SDK\Include

File	Description
`SDK10000.h`	SDK 10000 include file.

Library Files ${SDK DIR}\SDK\LIB

File	Description
`KMpeg4.lib`	SDK 10000 library file.
`PTZParser.lib`	PTZ command parser.

Runtime DLL Files ${SDK DIR}\SDK\DLL

File	Description
`KMpeg4.dll`	SDK Kernel dll.
`AADP.dll`	
`(ATCP10.dll`	AVC adaptor on networking module for TCP 10 data.
`ATCP20.dll`	AVC adaptor on networking module for TCP 20 data.
`AMCST10.dll`	AVC adaptor on networking module for Multicast 10 data.
`AMCST20.dll`	AVC adaptor on networking module for Multicast 20 data.
`ARTP.dll`	AVC adaptor on networking module for RTP data.
`ARAW.dll)`	AVC adaptor for playback.
`ADADP.dll`	GDI viewer
`(DGDI.dll`	
`DxDraw.dll)`	DirectX viewer
`AFADP.dll`	
`(FRAW.dll`	File adaptor on raw data format
`FAVI.dll)`	File adaptor on avi data format
`FFMCODEC.dll`	MPEG-4 software CODEC
`XVIDCODEC.dll`	MPEG-4 software CODEC

`P51CODEC.dll`	MPEG-4 software CODEC
`WISCODEC.dll`	MPEG-4 software CODEC
`IH264CODEC.dll`	H.264 software CODEC
`IPPCodec.dll`	IPP CODEC
`ipp*.dll`	IPP related functions
`MJPEGCODEC.dll`	Motion JPEG software CODEC
`PTZParser.dll`	PTZ supporting functions

Sample Codes ${SDK DIR}\SDK\Samples

SDK-10000 v1.2 sample programs can be reached at **${SDK Directory}\SDK\Samples**

SDK-10000 v1.2 provides several samples:

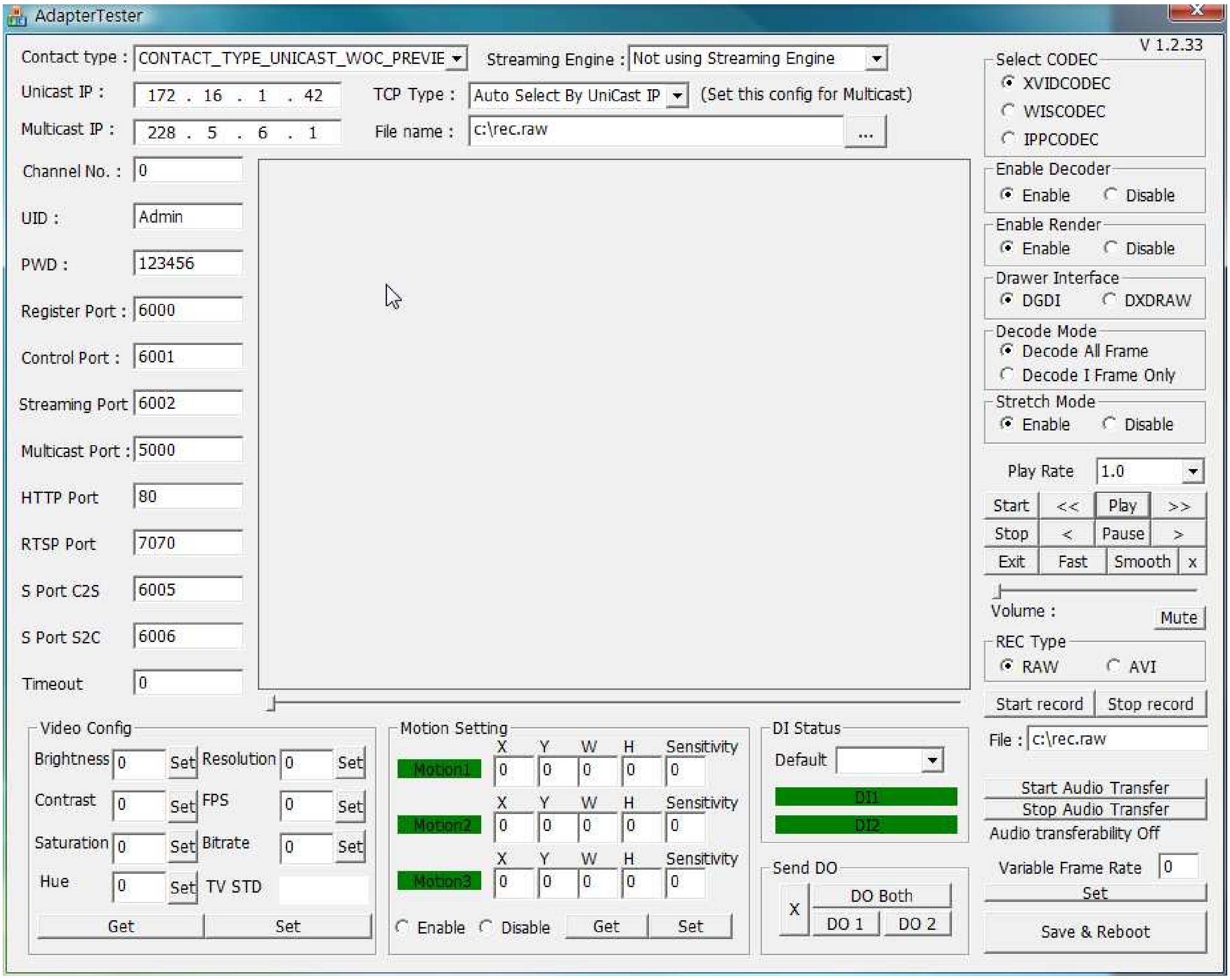
C Edition Sample Codes					
Sample Code	**Description**	VC6	VC2003	VC2005	VC2008
AbsolutePosition	Use absolute position to control PTZ Camera.	●	●	●	●
ArchivePlayer	Preview a RAW file with playback function.		●	●	●
ControlSample	Setup control port connection and receive event from device directly	●	●	●	●
DecodeSample	Connects to the device receives media raw data decode MPEG4/H.264 to RGB buffer display RGB buffer Save to BMP file		●	●	●
MediaConverter	Convert RAW file to AVI format		●	●	●
PTZSample	Get PTZ command from PTZParser library Send PTZ command via URL command	●	●	●	●
RTPSample	Connects to device using RTP over UDP or RTP over Multicast Audio supported		●	●	●
StreamSample	Live view via TCP, Multicast, RTP 2-way audio, Record, Playback Motion Detection, DIO Get/Set device configuration	●	●	●	●
SearchSample	Search for connectable devices	●	●	●	●
SendAudio	Send wave file to device.	●	●	●	●
URLSample	Send URL request and receive URL	●	●	●	●

	response from device.				

StreamSample Program

StreamSample codes demonstrate following functions:

1. Search Server
2. Connection mode: unicast, multicast
3. Preview, Record
4. Motion Detection set up and trigger
5. DI trigger and sends DO
6. Audio functions


AdapterTester

V 1.2.33

Contact type : CONTACT_TYPE_UNICAST_WOC_PREVIE
Streaming Engine : Not using Streaming Engine
Unicast IP : 172 . 16 . 1 . 42
TCP Type : Auto Select By UniCast IP (Set this config for Multicast)
Multicast IP : 228 . 5 . 6 . 1
File name : c:\rec.raw
Channel No. : 0
UID : Admin
PWD : 123456
Register Port : 6000
Control Port : 6001
Streaming Port 6002
Multicast Port : 5000
HTTP Port 80
RTSP Port 7070
S Port C2S 6005
S Port S2C 6006
Timeout 0

Select CODEC: ● XVIDCODEC ○ WISCODEC ○ IPPCODEC
Enable Decoder: ● Enable ○ Disable
Enable Render: ● Enable ○ Disable
Drawer Interface: ● DGDI ○ DXDRAW
Decode Mode: ● Decode All Frame ○ Decode I Frame Only
Stretch Mode: ● Enable ○ Disable
Play Rate 1.0
REC Type: ● RAW ○ AVI
File : c:\rec.raw
Audio transferability Off
Variable Frame Rate 0

Video Config
Brightness 0
Contrast 0
Saturation 0
Hue 0
Resolution 0
FPS 0
Bitrate 0
TV STD

Motion Setting

	X	Y	W	H	Sensitivity
Motion1	0	0	0	0	0
Motion2	0	0	0	0	0
Motion3	0	0	0	0	0

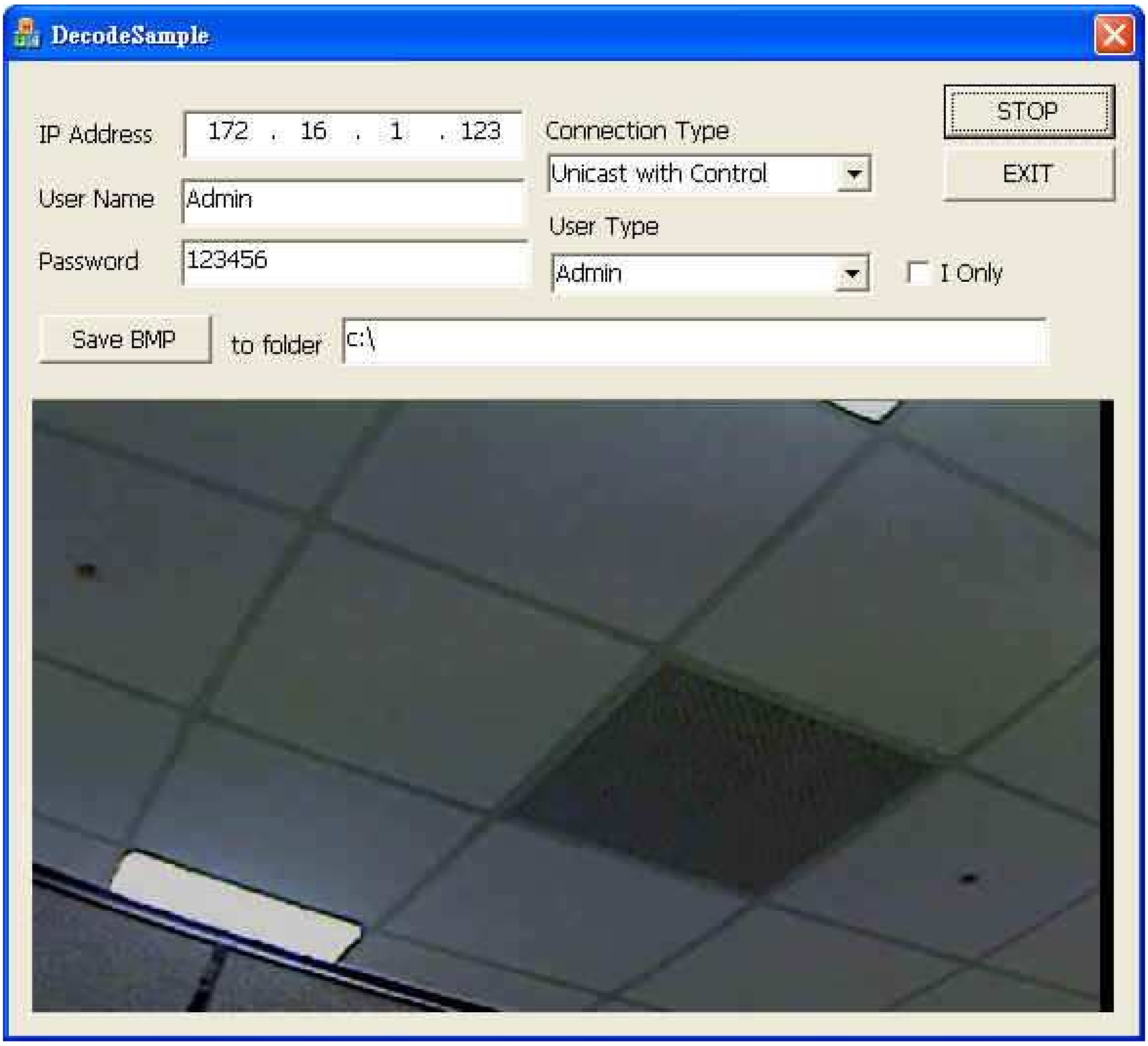
○ Enable ○ Disable

DI Status
Default

DecodeSample Program

PlaybackSample codes demonstrate following functions:

1. Decode MPEG-4/H.264 into RGB buffer
2. Display RGB buffer onto screen
3. Save RGB buffer to BMP file
4. Decode I-frame only

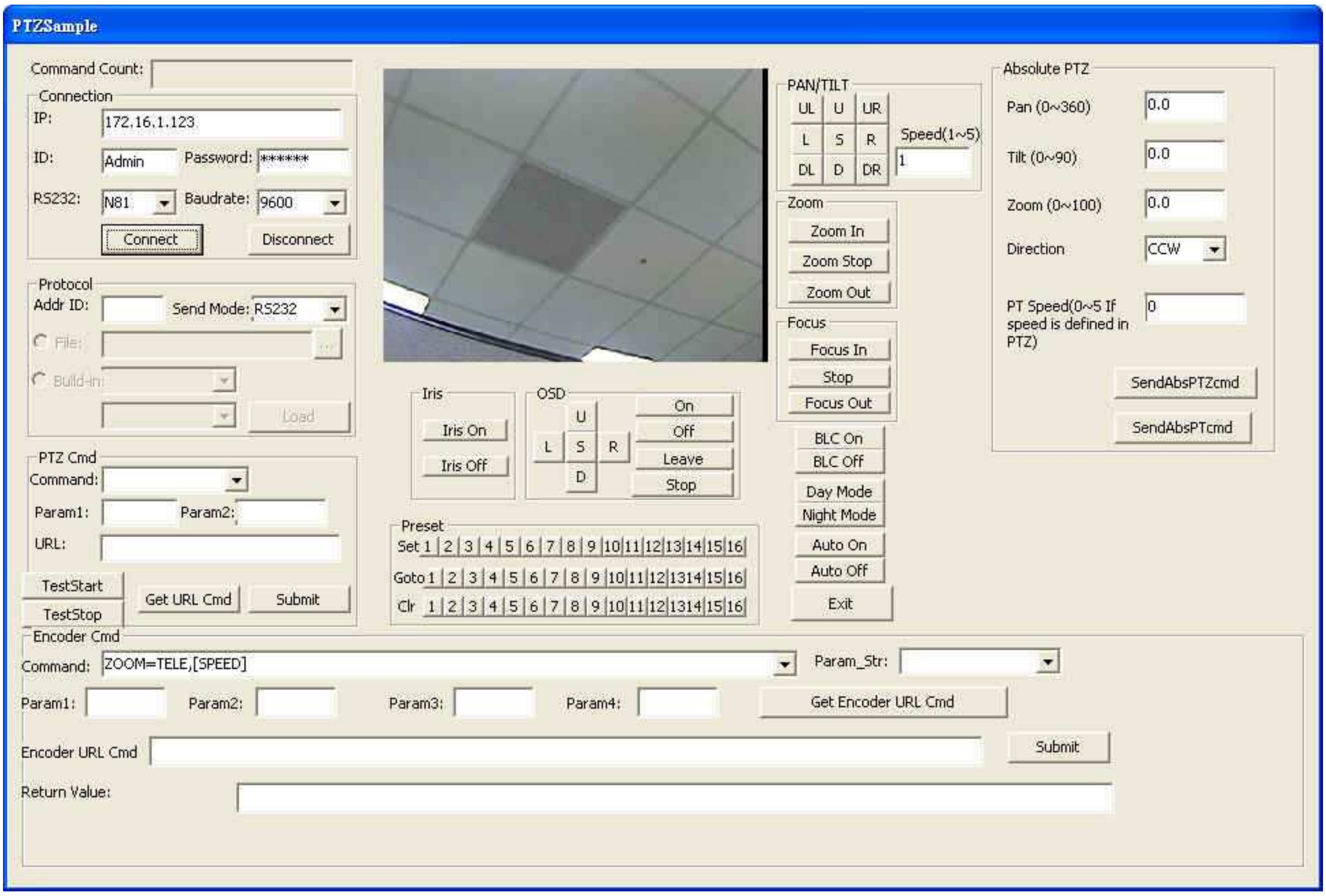


IP Address: 172 . 16 . 1 . 123
User Name: Admin
Password: 123456
Connection Type: Unicast with Control
User Type: Admin
☐ I Only
to folder: c:\

PTZSample Program

PTZSample codes demonstrate following functions:

1. Read PTZ protocol files
2. Operate PTZ functions.(Most PTZ functions were updated since V1.2)
3. Demonstrate Pan, Tilt, Zoom, Focus, Iris, Preset, OSD, and Absolute PTZ functions. (Absolute PTZ functions only work with DynaColor protocols now.)
4. URL Command to send PTZ commands.
5. Get PTZ command using PTZParser library. (PTZParser was integrated into SDK V1.2, so that is major change of PTZ APIs.).



PTZSample

IP: 172.16.1.123

ID: Admin

Password: ******

RS232: N81

Baudrate: 9600

Send Mode: RS232

Speed(1~5): 1

Pan (0~360): 0.0

Tilt (0~90): 0.0

Zoom (0~100): 0.0

Direction: CCW

PT Speed(0~5 If speed is defined in PTZ): 0

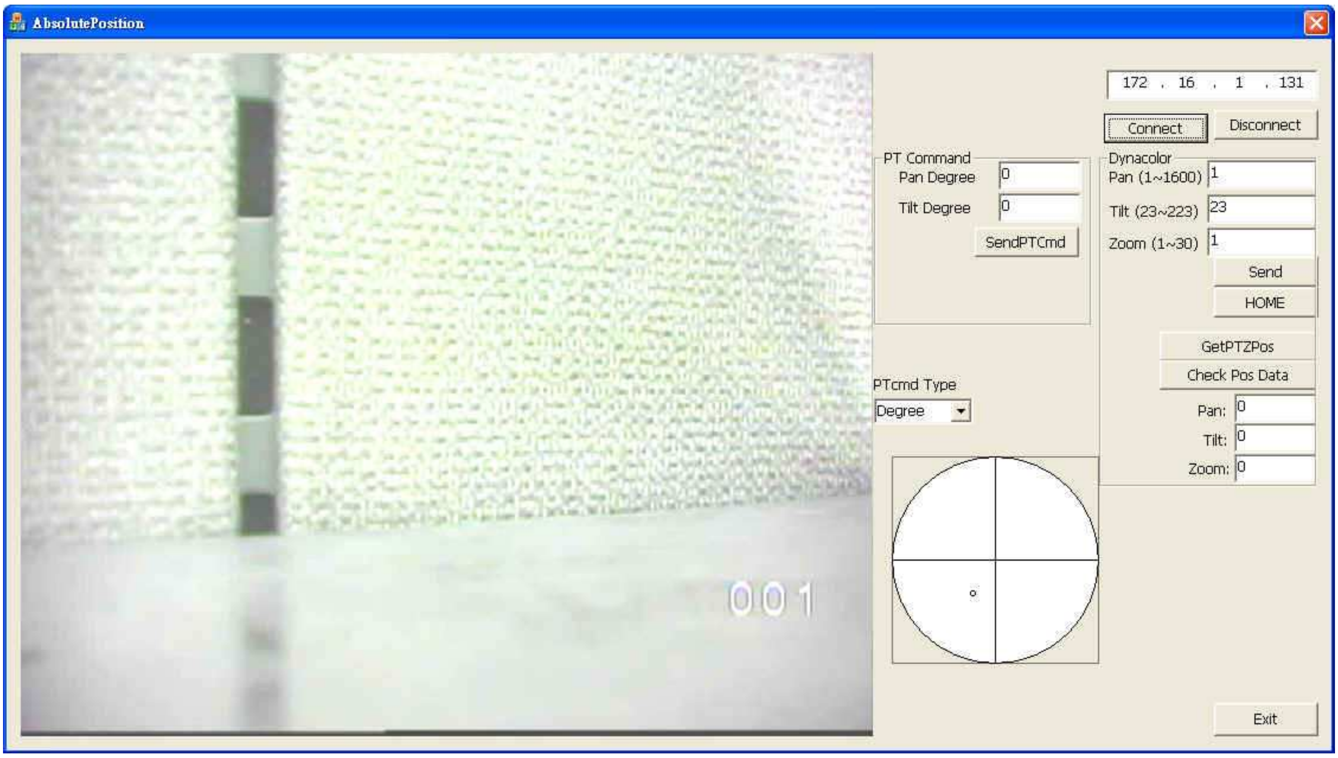
Command: ZOOM=TELE,[SPEED]

AbsolutePosition Program

AbsolutePosition codes demonstrate following functions:

1. Read PTZ protocol files
2. Operate PTZ functions.(Demonstrate Pan, Tilt, and Zoom)
3. Get current PTZ position.

(This example works with Dynacolor protocol only.)



172 . 16 . 1 . 131

Pan Degree 0

Tilt Degree 0

Pan (1~1600) 1

Tilt (23~223) 23

Zoom (1~30) 1

PTcmd Type Degree

Pan: 0

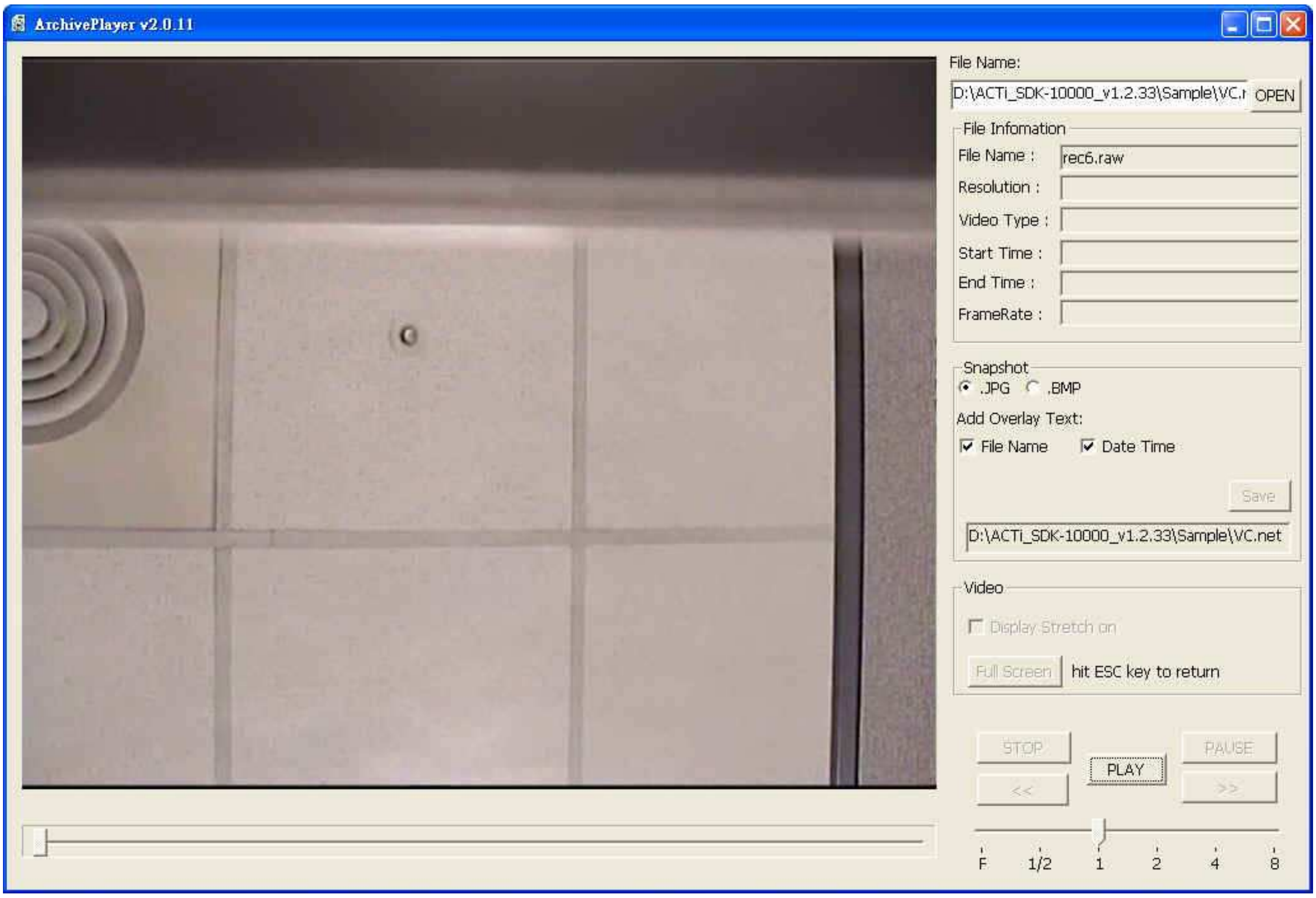
Tilt: 0

Zoom: 0

ArchivePlayer Program

ArchivePlayer codes demonstrate following functions:

1. Snapshot with JPG&BMP format.
2. Play with different speed.
3. Preview with frame by frame.
4. Pause.
5. Seek into random position.
6. Allow to play raw/mp4 file.
7. Display text on video frame.



File Name:
D:\ACTi_SDK-10000_v1.2.33\Sample\VC.r

File Infomation
File Name : rec6.raw
Resolution :
Video Type :
Start Time :
End Time :
FrameRate :

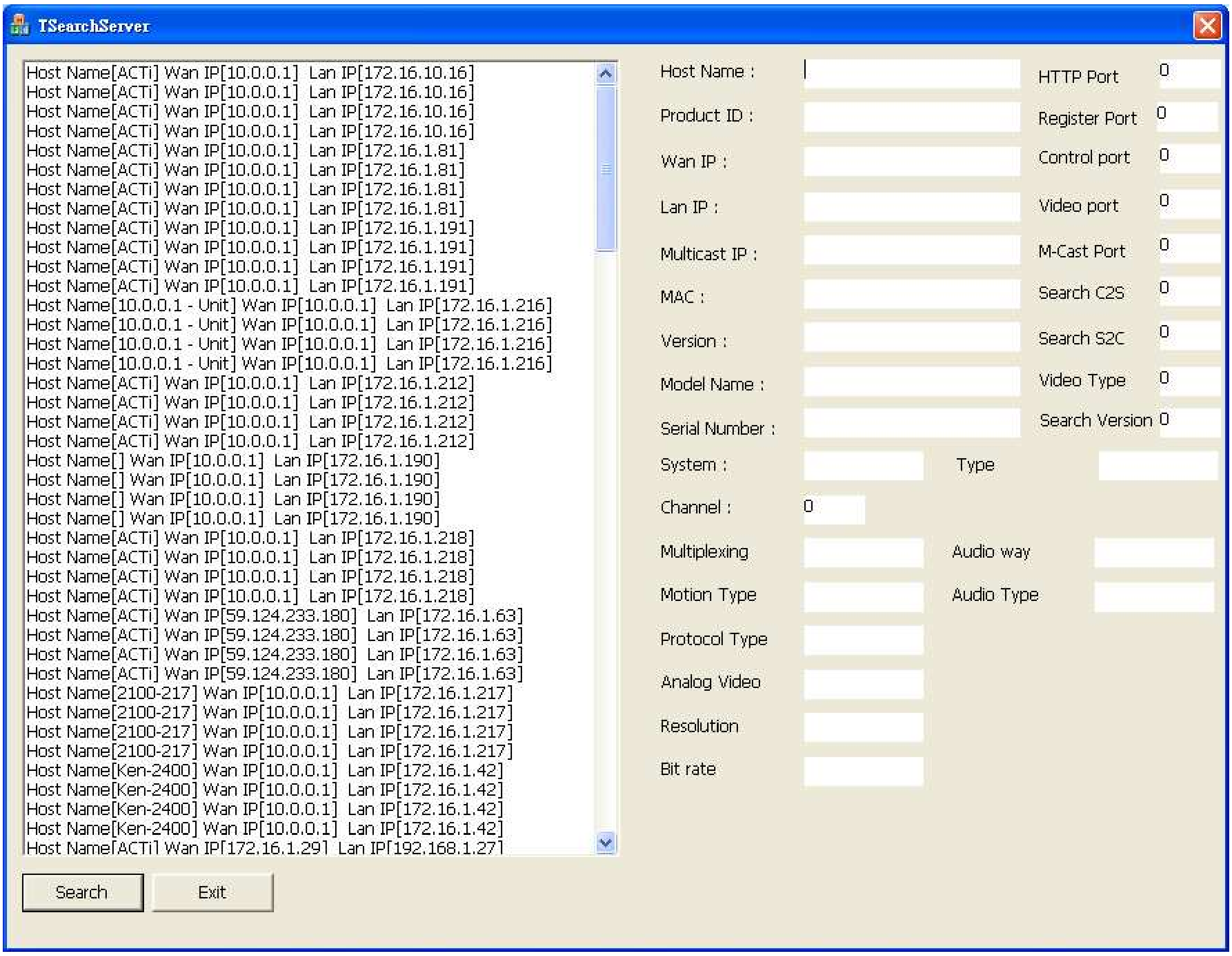
Snapshot
● .JPG ○ .BMP
Add Overlay Text:
☑ File Name ☑ Date Time
D:\ACTi_SDK-10000_v1.2.33\Sample\VC.net

Video
☐ Display Stretch on
hit ESC key to return

SearchSample Program

SearchSample codes demonstrate following functions:

1. Search for connectable devices..



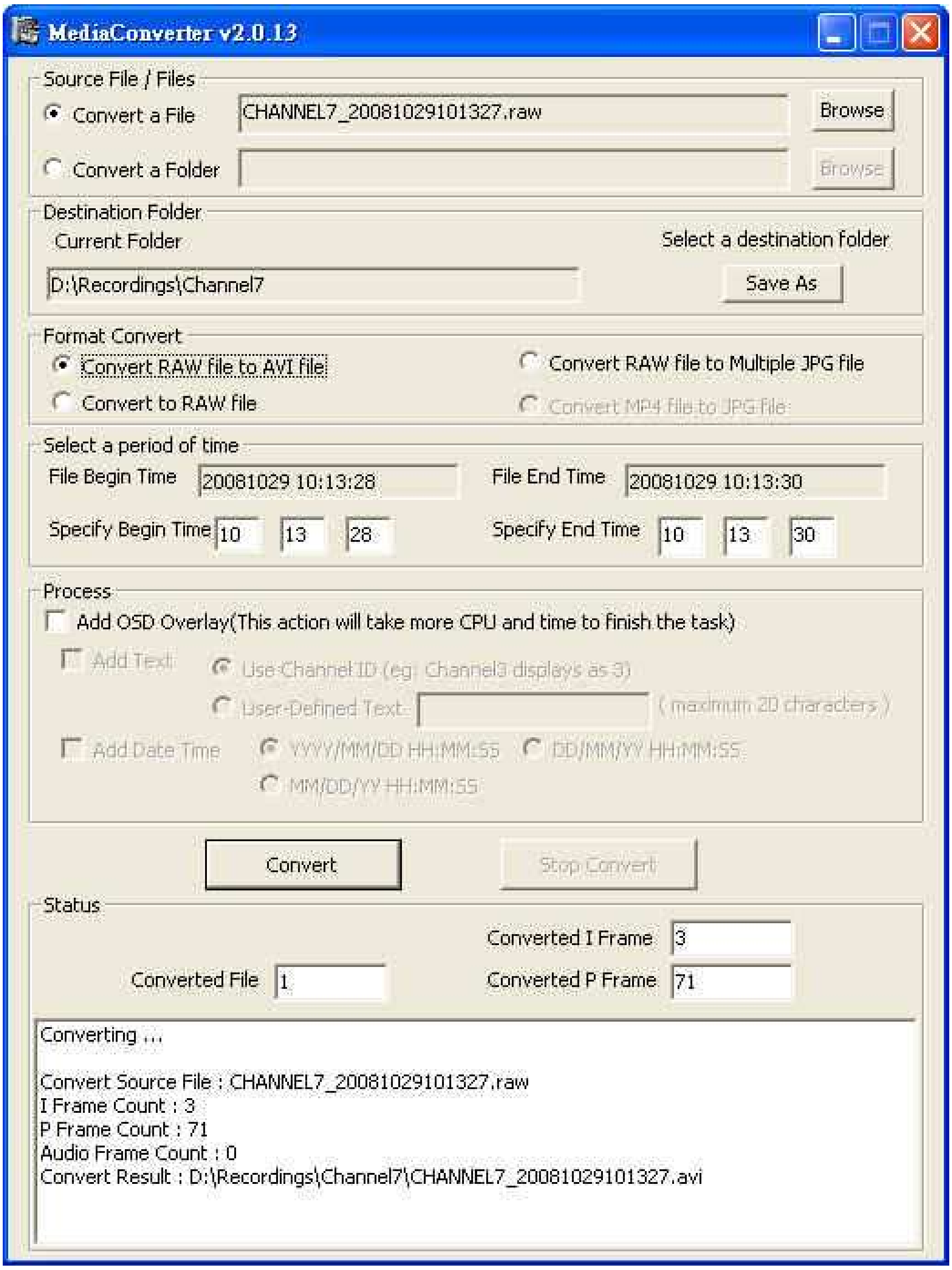
```
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.10.16]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.10.16]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.10.16]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.10.16]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.81]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.81]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.81]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.81]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.191]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.191]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.191]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.191]
Host Name[10.0.0.1 - Unit] Wan IP[10.0.0.1]  Lan IP[172.16.1.216]
Host Name[10.0.0.1 - Unit] Wan IP[10.0.0.1]  Lan IP[172.16.1.216]
Host Name[10.0.0.1 - Unit] Wan IP[10.0.0.1]  Lan IP[172.16.1.216]
Host Name[10.0.0.1 - Unit] Wan IP[10.0.0.1]  Lan IP[172.16.1.216]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.212]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.212]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.212]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.212]
Host Name[] Wan IP[10.0.0.1]  Lan IP[172.16.1.190]
Host Name[] Wan IP[10.0.0.1]  Lan IP[172.16.1.190]
Host Name[] Wan IP[10.0.0.1]  Lan IP[172.16.1.190]
Host Name[] Wan IP[10.0.0.1]  Lan IP[172.16.1.190]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.218]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.218]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.218]
Host Name[ACTi] Wan IP[10.0.0.1]  Lan IP[172.16.1.218]
Host Name[ACTi] Wan IP[59.124.233.180]  Lan IP[172.16.1.63]
Host Name[ACTi] Wan IP[59.124.233.180]  Lan IP[172.16.1.63]
Host Name[ACTi] Wan IP[59.124.233.180]  Lan IP[172.16.1.63]
Host Name[ACTi] Wan IP[59.124.233.180]  Lan IP[172.16.1.63]
Host Name[2100-217] Wan IP[10.0.0.1]  Lan IP[172.16.1.217]
Host Name[2100-217] Wan IP[10.0.0.1]  Lan IP[172.16.1.217]
Host Name[2100-217] Wan IP[10.0.0.1]  Lan IP[172.16.1.217]
Host Name[2100-217] Wan IP[10.0.0.1]  Lan IP[172.16.1.217]
Host Name[Ken-2400] Wan IP[10.0.0.1]  Lan IP[172.16.1.42]
Host Name[Ken-2400] Wan IP[10.0.0.1]  Lan IP[172.16.1.42]
Host Name[Ken-2400] Wan IP[10.0.0.1]  Lan IP[172.16.1.42]
Host Name[Ken-2400] Wan IP[10.0.0.1]  Lan IP[172.16.1.42]
Host Name[ACTi] Wan IP[172.16.1.29]  Lan IP[192.168.1.27]
```

HTTP Port 0
Register Port 0
Control port 0
Video port 0
M-Cast Port 0
Search C2S 0
Search S2C 0
Video Type 0
Search Version 0
Channel : 0

MediaConverter Program

MediaConverter codes demonstrate following functions:

1. Convert raw file to avi or multiple jpg.



Source File / Files

● Convert a File: CHANNEL7_20081029101327.raw

○ Convert a Folder:

Destination Folder

Current Folder: D:\Recordings\Channel7

Select a destination folder

Format Convert

● Convert RAW file to AVI file

○ Convert RAW file to Multiple JPG file

○ Convert to RAW file

○ Convert MP4 file to JPG file

Select a period of time

File Begin Time: 20081029 10:13:28

File End Time: 20081029 10:13:30

Specify Begin Time: 10 13 28

Specify End Time: 10 13 30

Process

☐ Add OSD Overlay(This action will take more CPU and time to finish the task)

☐ Add Text ● Use Channel ID (eg: Channel3 displays as 3)

○ User-Defined Text: (maximum 20 characters)

☐ Add Date Time ● YYYY/MM/DD HH:MM:SS ○ DD/MM/YY HH:MM:SS

○ MM/DD/YY HH:MM:SS

Status

Converted I Frame: 3

Converted File: 1

Converted P Frame: 71

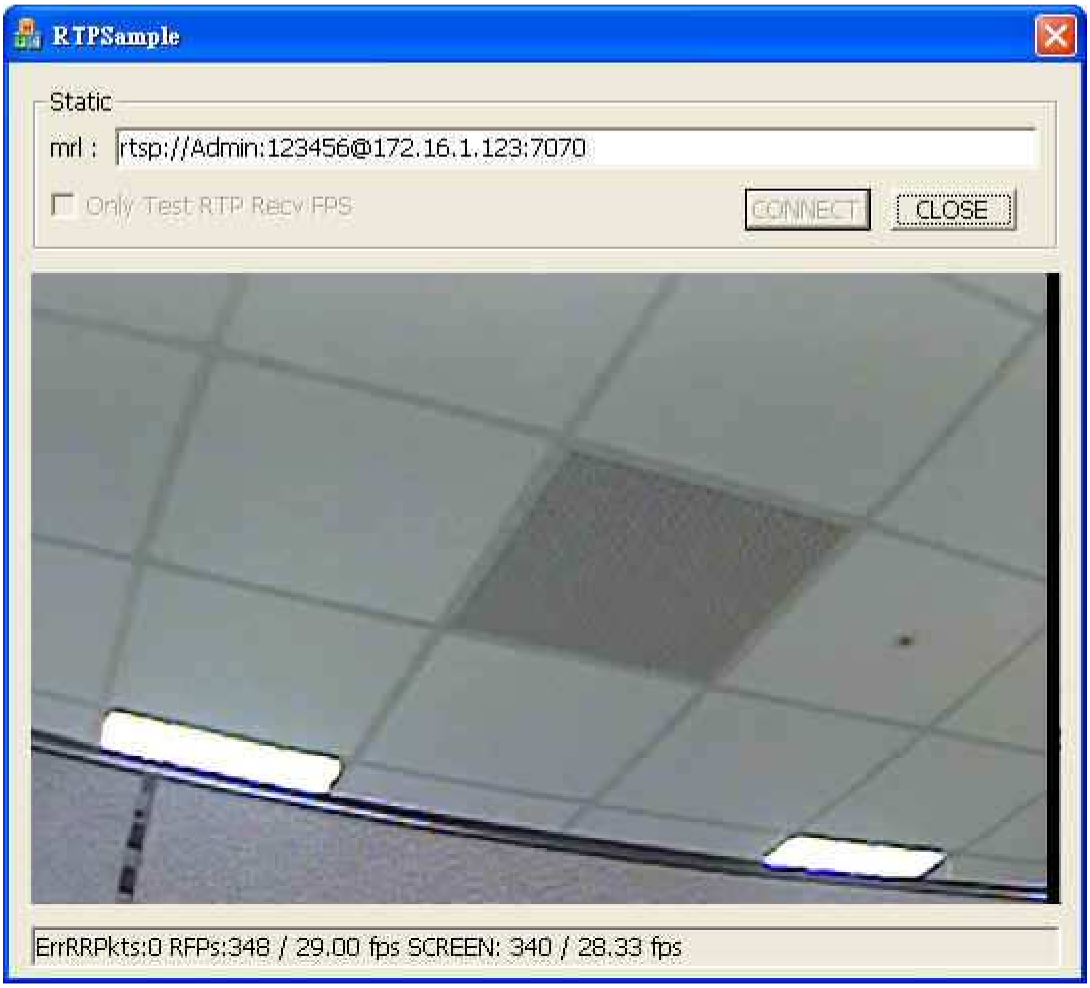
```
Converting ...

Convert Source File : CHANNEL7_20081029101327.raw
I Frame Count : 3
P Frame Count : 71
Audio Frame Count : 0
Convert Result : D:\Recordings\Channel7\CHANNEL7_20081029101327.avi
```

RTPSample Program

RTPSample codes demonstrate following functions:

1. Connect camera by RTP and RTSP
2. Show up RTP and RTSP transferring state.
3. Enabling “Only Test RTP Recv FPS” option will skip all pocket process.

ErrRRPKts : Show error pocket number. (Including missed and wrong sequence)
RFPs : Received frames number by socket per second.
SCREEN : Processed frames number in a second.



mrl : rtsp://Admin:123456@172.16.1.123:7070

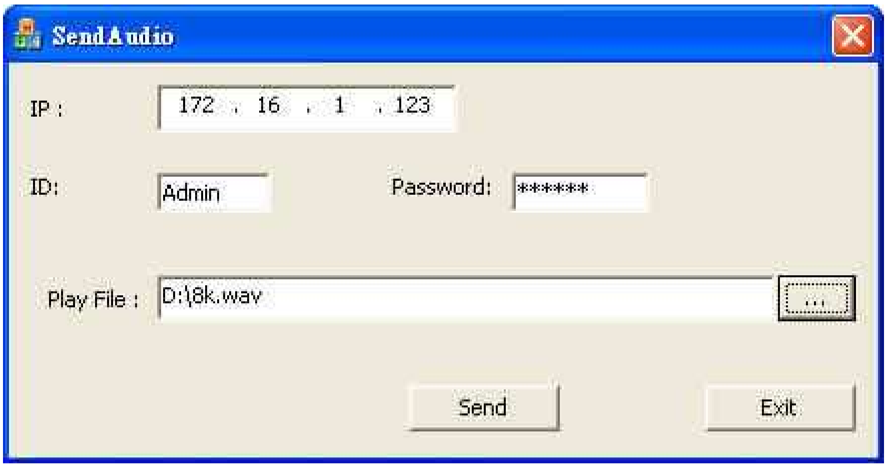
☐ Only Test RTP Recv FPS

```
ErrRRPkts:0 RFPs:348 / 29.00 fps SCREEN: 340 / 28.33 fps
```

SendAudio Program

SendAudio codes demonstrate following functions:

1. Select a pcm of 8k wave file.
2. Press send button to send audio to device.



IP : 172 . 16 . 1 . 123

ID: Admin

Password: ******

Play File : D:\8k.wav

2 Search Device

Device Locator Architecture

The section describes the mechanism on how to search IP surveillance products on network. With this mechanism, you can locate the devices on the network, then use URL commands to operate or manage those devices.

The function sends out a broadcast message, devices respond with detailed information, application then parse the replied information and parse the content with **NET_SEARCHSERVER** data structure.

Search Device

Steps to detect IP Surveillance products are listed as follow:

1. Call `netSearchServer()`
2. Receive and decodes with **NET_SEARCHSERVER**

> **NOTE:** The second parameter of `netSearchServer()` indicates the maximum total number to be reached in the network; for example, if this parameter is set to 10, and there are 20 devices in the same network, then this function returns when it reaches the first 10 devices in the network.
>
> Default timeout value is 20 seconds

```
typedef struct tagSearchServer {
 char szHostName[24];       // [OUT] Host Name         : ASCII Z STRING
 char szProductID[8];       // [OUT] ProductID         : ASCII Z STRING
 char szWanIp[16];          // [OUT] WAN IP            : ASCII Z STRING
 char szLanIp[16];         // [OUT] LAN IP        : ASCII Z STRING
 char szMultiCastIp[16];    // [OUT] MULTICAST IP      : ASCII Z STRING
 char szMac[32];            // [OUT] MAC           : ASCII Z STRING
 char cType;                // [OUT] Bit0~3        : 1: Composite, 2: S-Video
                            // [OUT] Bit4~7      : 1: Video Server, 2: IPCam
 char  dummy1;
 char  dummy2;
 char  dummy3;
char  Version[32];
 WORD  wHPort;
 WORD  wSPortC2S;           // [IN]  Search   Port (Client to Server)
 WORD  wSPortS2C;           // [IN]  Search   Port (Server to Client)
 WORD  wRPort;              // [IN]  Register Port
 WORD wCPort;               // [IN]  Control  Port
 WORD wVPort;               // [IN]  Video    Port
 WORD wMPort;               // [IN]  MultiCastPort
 WORD dummy4;

} NET_SEARCHSERVER;



WORD dwRet ;
NET_SEARCHSERVER ServerList[MAXSERVERLIST];
 // Receive data Structure

DWORD dwTotalNum = MAXSERVERLIST ;

dwRet = netSearchServer((char*) ServerList, &dwTotalNum);

for (DWORD i = 0; i< dwTotalNum; i++) {
 szHostName[i]     = ServerList[i].szHostName ;
     // Get the Host Name From Result Structure
 szProductID[i]        = ServerList[i].szProductID ;
     // Get the Product ID From Result Structure
 szWanIp[i]            = ServerList[i].szWanIp ;
     // Get the WanIp From Result Structure
 szLanIp[i]            = ServerList[i].szLanIp
     // Get the LanIp From Result Structure
 szMultiCastIp[i]  = ServerList[i].szMultiCastIp ;
     // Get the MultiCastIp From Result Structure
 szMac[i]          = ServerList[i].szMac ;
     // Get the Mac Address From Result Structure
 szVersion[i]      = ServerList[i].Version ;
     // Get the Firmware Version From Result Structure
 wRPort[i]             = ServerList[i].wRPort;
     // Get the Register Port From Result Structure
 wCPort[i]             = ServerList[i].wCPort;
     // Get the control Port From Result Structure
 wVPort[i]             = ServerList[i].wVPort;
     // Get the Streaming Port From Result Structure
 wMPort[i]             = ServerList[i].wMPort;
     // Get the Multicast Port From Result Structure
 wHPort[i]             = ServerList[i].wHPort;
     // Get the Http Port From Result Structure
}
```

How to detect device

This section describes how to detect, manage and configure IP devices. All commands are operated with URL Commands, you can use the functions we suggested (xmlhttp) or you can find HTTP-related functions by yourselves.

Please also refer to the Appendix for the complete URL Command listing.

System Information

Steps to detect product System Information are listed as follow:

Sample:

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
 KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

strURL = 'http://192.168.1.100:80' ;
strURL = '/cgi-bin/system?USER=Admin&PWD=123456&SYSTEM_INFO' ;

 char szResultBuf[1024] = {0};
 DWORD dwResultLen;
 KSendURLCommand( hK, strURL, szResultbuf, dwResultLen) ;

//    Firmware Version = A1D-M2N-V2.03.02-NB
//    MAC Address = 00:0F:7C:00:1A:47
//    Production ID = SED2400-05I-1-00034
//    Factory Default Type = NTSC, Composite, Two Ways Audio (0x71)
```

System Property

Steps to detect product System Property are listed as follow:

Sample:

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
KsetMediaConfig2(hK, &mcc);
KConnect(hK);

strURL = 'http://192.168.1.100:80' ;
strURL = '/cgi-bin/system?USER=Admin&PWD=123456& SYSTEM_PROPERTY ' ;

 char szResultBuf[1024] = {0};
 DWORD dwResultLen;
 KSendURLCommand( hK, strURL, szResultbuf, dwResultLen) ;

//    SYSTEM='E'
//    TYPE='A'
//    NO_OF_CHANNEL='01'
//    MULTIPLEXING='X'
//    NO_OF_AUDIO_WAYS='2'
//    AUDIO_TYPE='PCM'
//    MOTION_TYPE='0'
//    PROTOCOL_TYPE='2'
```

Video Color Adjustments

This section describes on how to adjust video color using URL Commands.

Hue, Brightness, Contrast Setting

Steps to Gets/Sets product Video Property are listed as follow:

1. Initial **`KMpeg4`** Object
2. Gets color setting.
3. Set new setting

Sample:

```
typedef struct structural_MEDIA_VIDEO_CONFIG2
{
   short dwEncoder;         // 1:MPEG4 4:MPEG4 5:H264
   short dwTvStander;           // 0:NTSC 1:PAL
   short dwVideoResolution; // See the definition above
   short dwBitsRate;            // See the definition above
   short dwQuality;         // 0 ~ 100 : Low ~ High
   short dwVideoBrightness; // 0 ~ 100 : Low ~ High
   short dwVideoContrast;       // 0 ~ 100 : Low ~ High
   short dwVideoSaturation; // 0 ~ 100 : Low ~ High
   short dwVideoHue;            // 0 ~ 100 : Low ~ High
   short dwFps;                 // 0 ~ 30 frame pre second
} MEDIA_VIDEO_CONFIG2;

// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …

KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

// Get current color setting
 MEDIA_VIDEO_CONFIG2 mvc;
 KgetVideoConfig2(hK, &mvc);
```

```
//    To Set the Video Property
 KSetHue(hK, 10)
 KSetBrightness(hK, 20);
 KSetContrast(hK, 30);
```

Video Setting Configuration

Setup Resolution, Frame Rate, Bit Rate

Steps to Gets/Sets product Video Setting are listed as follow:

Sample:

```
enum BITRATE_TYPES /** Bitrate Types */
{
BITRATE_28K,            ///< #0# - 28K Bits per second
BITRATE_56K,            ///< #1# - 56K Bits per second
…
BITRATE_6000K,          ///< #12# - 6M Bits per second
}

enum RESOLUTION_TYPES   /** Resolution Types */
{
NTSC_720x480,           ///< #0# - NTSC - 720 x 480
NTSC_352x240,           ///< #1# - NTSC - 352 x 240
…
PAL_176x144             ///< #5# - PAL  - 176 x 144
}

typedef struct structural_MEDIA_VIDEO_CONFIG2
{
   short dwEncoder;         // 1:MPEG4 4:MPEG4 5:H264
   short dwTvStander;           // 0:NTSC 1:PAL
   short dwVideoResolution; // See the definition above
   short dwBitsRate;            // See the definition above
   short dwQuality;         // 0 ~ 100 : Low ~ High
   short dwVideoBrightness; // 0 ~ 100 : Low ~ High
   short dwVideoContrast;       // 0 ~ 100 : Low ~ High
   short dwVideoSaturation; // 0 ~ 100 : Low ~ High
   short dwVideoHue;            // 0 ~ 100 : Low ~ High
   short dwFps;                 // 0 ~ 30 frame pre second
} MEDIA_VIDEO_CONFIG2;

// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
KsetMediaConfig2(hK, &mcc);
```

```
 KConnect(hK);

// Get current color setting
 MEDIA_VIDEO_CONFIG2 mvc;
 KGetVideoConfig2(hK, &mvc);

//    To Set the Video Property
 KSetResolution(hK, 10)     // 0~5
 KSetFPS(hK, 30);
 KSetBitRate(hK, 30);       // 0~12
```

Save and Reboot

The section describes the mechanism on how to search IP surveillance products on network. With this mechanism, you can locate the devices on the network, then use URL commands to operate or manage those devices.

The function sends out a broadcast message, devices respond with detailed information, application then parse the replied information and parse the content with **NET_SEARCHSERVER** data structure.

Execute Save and Reboot Command

Steps to execute Save and Reboot Video device are listed as follow:

Sample:

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

KSaveReboot(hK);
```

3

Preview / Record / Playback

Preview / Record Architecture

This material covers SDK architecture, data structures and sample programs to illustrate the methods to integrate IP Surveillance products.

Register to IP devices

Steps to register to device:

1. Call `KOpenInterface()` to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Call `KSetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)` to set connect config.
4. Call `KConnect(HANDLE).`
5. Call `KStartStreaming(HANDLE)` to get ready to receive.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
KsetMediaConfig2(hK, &mcc);
KConnect(hK);

// Start Streaming
 KStartStreaming(hK);
```

Dual stream devices and multi channel devices

Choose stream or channel number

Steps to register to device:

1. Call **`KOpenInterface()`** to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Set ChannelNumber in MEDIA_CONNECTION_CONFIG2 structure.
4. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
5. Call **`KConnect(HANDLE).`**
6. Call **`KStartStreaming(HANDLE)`** to get ready to receive.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ChannelNumber = 3; //(Select channel no.4 in a multi channel device )
 …
KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);
```

Variable Frame Rate and Multi-Stream

Choose stream or channel number

When the device is set on variable frame rate mode, the device is able to send variable frame rate with different TCP session.

Create multiple handles to connect devices:

1. Call **`KOpenInterface()`** to get KMpeg4 handles.
2. Prepare IP address, port number, account, password, contact type..
3. Set ChannelNumber in MEDIA_CONNECTION_CONFIG2 structure.
4. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
5. Call **`KConnect(HANDLE).`**
6. Call **`KStartStreaming(HANDLE)`** to get ready to receive.

```
// you should get HANDLEs by KOpenInterface
HANDLE hK1 = KOpenInterface();
HANDLE hK2 = KOpenInterface();


// Prepare USER_INFO data structure by filling IP address, account, password.

// Set your connection information into struct mcc.
 …

// Start Streaming
 KStartStreaming(hK1);
 Kplay(hk2);

 KSetVariableFPS(hk1, 1);
 KSetVariableFPS(hk2, 30);
```

Preview Operations

Preview with Unicast Mode

Steps to start preview with unicast mode include:

1. Set contact type as **`CONTACT_TYPE_UNICAST_PREVIEW`**;
2. Register to the IP devices
3. Call **`KPlay(HANDLE)`** to start receive data.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK =  KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_UNICAST_PREVIEW;
 …
KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);
// Start receiving data from KMpeg4
 KPlay(hK);
```

Preview with Audio

Steps to register to device:

1. Call **`KOpenInterface()`** to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
4. Call **`KConnect(HANDLE).`**
5. Call **`KStartStreaming(HANDLE)`** to get ready to receive.
6. Call **`KPlay(HANDLE)`** to start receive data.
7. Set mute mode to false with **`KSetMute(HANDLE, BOOL)`** function
8. Set audio volume with **`KSetVolume(HANDLE, int, int)`** function

NOTE:

```
    /
// Register to the device
    // Start Preview

    //---- Set volume
    KSetVolume( hK , lLeftVolume , lReightVolume );   // set volume

    //---- set to mute
    KSetMute(hK, true);          // audio is off

    //---- turn audio back on
    KSetMute(hK, false);         // audio is on
```

Preview with 2-way audio

Steps to preview with 2-way audio include:

1. Call `KOpenInterface()` to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Call `KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)` to set connect config.
4. Call `KConnect(HANDLE)`.
5. Call `KStartStreaming(HANDLE)` to get ready to receive.
6. Call `KPlay(HANDLE)` to start receive data.
7. Start preview
8. Get Audio Token
9. Send audio sound from PC side to the device with `KStartAudioTransfer(HANDLE)` function. This function opens the speaker connected on the PC, and grab sound from the speaker and transmit to the device
10. Stop sending audio sound from PC side to the device with `KStopAudioTransfer(HANDLE)` function

IMPORTANT: One IP device has only 1 audio token; if the token is taken by one application, then no other application may acquire the audio token again. Remember to free audio token after the 2-way audio function is done.

```
// Register to the device

// Get the Audio Token
bool bAudioToken = KGetAudioToken( hK );

// check the return value , if you get the audio token success.
if ( bAudioToken )
{
      KStartAudioTransfer(hK);
// start sending audio from PC to the device
// this function turns on speaker, the audio will be captured
// and transferred to the devices
}
KStopAudioTransfer(hK);
// Free the Audio Token Before you close connection.
KFreeAudioToken(hK);
```

Preview with I-Frame Decoding only

This chapter describes a mechanism on how to decrease CPU loading. With this mechanism, MPEG-4 software decoder will decode I-Frame only and drops all P-Frame before decoding.

Steps to preview with I-Frame decoding only include:

1. Register to the IP device
2. Preview with **`KPlay(HANDLE)`**
3. Set to I-Frame decoding only with **`KSetDecodeIFrameOnly(HANDLE, BOOL)`** function

NOTE: With **`KSetDecodeIFrameOnly(HANDLE, BOOL)`** function, the CPU loading can be decreased dramatically.

IMPORTANT: **`KSetDecodeIFrameOnly(HANDLE, BOOL)`** function only affects preview and CPU loading; recording still records with I-frame and P-frame as setup.

```
   // you should get HANDLE by KOpenInterface and Start Preview First
    KPlay(hK);

   // [1] If you are handling raw data yourself by using call back function then
you //      have to filter the frames and decide which frame your are going to process.
   //      This is because KMpeg4 will pass all the frames to call back function.

   // Determine the frame type I or P frame.

   If (!bDecodeI )
   {
         // Decode All of Frames you receive
    } else {
         // Check the frame type
         // Decode I Frame Only
    }

   //-------------------------------------------------------------------------
---

   // [2] If KMpeg4 is handling the raw data for you then you can call
   //      KSetDecodeIFrameOnly(HANDLE, BOOL) to decode I frame only
    KSetDecodeIFrameOnly(hK, true);
```

Draw your own information on the preview window

This chapter describes a mechanism on how you can draw your own information on the preview window, including OSD information, timecode or video intelligence information.

Steps to draw your own information on the preview window:

1. Register to the IP device
2. Setup after render callback function (**`KSetAfterRenderCallback()`**)
3. When preview window is painted, SDK will calls after render callback function
4. Draw your own information in the after render callback function

NOTE: When you hook up **`KSetAfterRenderCallback()`** function, the callback function will be called 30 times per second, if the frame rate is set to 30 FPS.

```
   /
// register to the device

   // Setup after render callback function
   KSetAfterRenderCallback( hK, dwCallbackID, AfterRenderCallback );


   AfterRenderCallback(DWORD dwCallbackID)
   {
   //---- draw your own information over here,
   //     including OSD, time code or video intelligence information
   }
```

Record Operations

Background record with multicast mode

Streaming Client Library is developed for Video Network Streaming Application.

Steps to start preview with multicast mode without preview include:

1. Set Contact type as **`CONTACT_TYPE_MULTICAST_PREVIEW`** or **`CONTACT_TYPE_MULTICAST_WOC_PREVIEW`**
2. Register to the IP devices
3. Start recording

NOTE: Application may start recording without preview.

```
/
// Get KMpeg4 handle
HANDLE hK =  KOpenInterface();

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_MULTICAST_PREVIEW;
 …
KsetMediaConfig2(hK, &mcc);
KConnect(hK);

// Start Streaming
 KStartStreaming(hK);
// Start receiving data from KMpeg4
 KPlay(hK);

// Start recording with record file name.
 KStartRecord(hK, “c:\\rec.raw”);

// Finish recording
// You can retrive the recording information by passing MP4FILE_RECORD_INFO
 MP4FILE_RECORD_INFO mri;
 KStopRecord(hK, &mri);
```

Alarm Recording with DI event

Steps to start alarm recording include:

1. Setup pre-event recording time and post-event recording time
2. Register to the IP devices
3. Setup event callback
4. Start alarm recording
5. Stop alarm recording

```
// Setup digital input callback function
KSetDICallback( hK, dwCallbackID, DIOCallback );

KSetPrerecordTime(hK, 5);   // set pre-event time as 5 seconds

// Register to the device

//----- in call back function

DIOCallback(DWORD dwCallbackID, bool bDI1, bool bDI2 )
{
 if ( bDI1 || bDI2)         //---- DI 1 or DI 2 is on
 {
     KStartRecord (hK, "C:\\AlarmREC.raw" );
     Sleep( 10000 );            // records for 10 seconds
     KStopRecord( hK, NULL );   // in total it records 15 seconds
 }
}
```

Playback Operations

Steps to operate playback functions include:

1. Call **`KOpenInterface()`** to get KMpeg4 handle.
2. Prepare file name and set contact type to **`CONTACT_TYPE_PLAYBACK`**
3. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
4. Call **`KConnect(HANDLE).`**
5. Call **`KStartStreaming(HANDLE)`** to get ready to receive.
6. Call **`KPlay(HANDLE)`** to start receive data.
7. Sets playback play speed
8. Calls playback operation, including play forward, play backward, seed operation

Open and close a raw data file

```
// Get KMpeg4 SDK handle
HANDLE hK =  KOpenInterface();

// Prepare playback file name.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_PLAYBACK;
 strcpy(mcc.PlayFileName, “c:\\test.raw”);
 …
KsetMediaConfig2(hK, &mcc);
// Open file.
 KConnect(hK);
// Start Streaming
 KStartStreaming(hK);


// Stop streaming
 KStopStream( hK );
// Close file
 KDisconnect( hK );
```

Play forward, backward

```
// Get KMpeg4 SDK handle
HANDLE hK =  KOpenInterface();

// Set render information.
 MEDIA_RENDER_INFO mri;
 mri.RenderInterface = DGDI;
 mri.hWnd = m_hWnd;               // Windows' handle to draw
 mri.rec = m_rec;                 // rec information.
 KSetRenderInfo(hK, &mri);

// Prepare playback file name.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_PLAYBACK;
 strcpy(mcc.PlayFileName, “c:\\test.raw”);
 …
KsetMediaConfig2(hK, &mcc);
// Open file.
 KConnect( hK );
// Start Streaming
 KStartStreaming(hK);
// Play forward
 KPlay( hK );

// Play backward
 KSetPlayDirection(hK, false);
```

Play frame by frame

```
// Play step by step

// Open file and play
   ...

// need to set play status pause for play step frame
     KPause(hK);

// Step to next frame
 KStepNextFrame(hK);

// Step to previous frame
 KStepPrevFrame(hK);
```

4 Event Handling

Digital I/O Architecture

This material covers SDK architecture, data structure and sample programs to illustrate the methods to integrate IP Surveillance products.

Receives Digital Input Event

Steps to receive digital input event include:

1. Register to the IP devices
2. Setup digital event callback
3. Process digital input event in the callback function

```
// Get Mpeg4 SDK handle
// Setup digital input callback function
KSetDICallback( hK, dwCallbackID, DICallback );

KSetPrerecordTime(hK, 5);   // set pre-event time as 5 seconds

// Register to the device

//----- in call back function

DICallback( DWORD dwCallbackID, bool bDI1, bool bDI2 )
{
 if ( bDI1 || bDI2 )        //---- DI 1 or DI 2 is on
 {
      KStartRecord (hK, "C:\\AlarmREC.raw" );
      Sleep( 10000 );            // records for 10 seconds
      KStopRecord( hK, NULL );   // in total it records 15 seconds
 }
}
```

Send Digital Output

Steps to receive digital input event include:

1. Register to the IP devices
2. Call **`KSendDO(HANDLE, BYTE)`** function to send event to the digital output device

Send DO 1

```
// Register to device.

#define DO_OUTPUT_1    0X01
#define DO_OUTPUT_2    0X02

// Send DO 1
 KSendDO( hK, DO_OUTPUT_1);
```

Send DO 2

```
// Register to device.

// Send DO 1
 KSendDO( hK, DO_OUTPUT_2);
```

Motion Detection Event Handling

Sets Motion Detection parameters

Steps to setup motion detection parameters include:

1. Register to the IP devices
2. Setup motion detection callback function
3. Sets motion detection parameters
4. Process motion detection event in the callback function

NOTE: The parameter to set the range of the motion detection window has to be the multiplier of 16, if not, the number will be aligning to the multiplier of 16. For example, if the application set the range as 125, then it will be align to 128.

Set MD Range to Range1

```
#define MD_REGION_SIZE 4
typedef struct structural_MEDIA_MOTION_INFO_EX
{
DWORD dwEnable;
DWORD dwRangeCount;
DWORD dwRange[MD_REGION_SIZE][4];
DWORD dwSensitive[MD_REGION_SIZE];
DWORD dwTime[MD_REGION_SIZE];
DWORD dwThreshold[MD_REGION_SIZE];
DWORD bEnable[MD_REGION_SIZE];
} MEDIA_MOTION_INFO_EX;

// Register to the IP devices

// Prepare you own callback function

// Plug function after KOpenInterface()
KSetMotionDetectionCallback2(hK, dwCallbackID, MDCallBack);

// Set motion detection structure
 MEDIA_MOTION_INFO_EX mmi;
 mmi.dwEnable = 1;                   // Enable MD
 mmri.dwRangeCount = 1;              // Just 1 range for MD
 mmi.dwSensitive[0] = 100;           // Sensitive of range 1
 mmi.dwRange[0][0] = 0;              // Left position
 mmi.dwRange[0][1] = 0;              // Top position
 mmi.dwRange[0][2] = 128;            // Width of range 1
 mmi.dwRange[0][3] = 128;            // Height of range 1
```

```
// Set motion detection information.
 KSetMotionInfoEx ( hK, mmi);
```

Gets Motion Detection Settings

Get MD Range Setting

```
//Prepare structure for get MD information
MEDIA_MOTION_INFO_EX mmi;

// One function to get all data
KGetMotionInfoEx(hK, &mmi);
```

Receives Motion Detection Trigger Event

To Plug You Own Callback Function for MD

```
void MDCallBack(DWORD dwCallbackID, unsigned char Motion, unsigned char PIR)
{
 if(Motion & 0x01)
 {
 // Motion 1 Event occuring
 }

 if(Motion & 0x02)
 {
 // Motion 2 Event occuring
 }

 if(Motion & 0x04)
 {
 // Motion 3 Event occuring
 }

 if(Motion & 0x08)
 {
 // Motion 4 Event occuring
 }

}
```

Status Callback – video lost, recovery, disconnect event

Status callback includes:

1. Video Lost event
2. Video Recorvery event
3. Network disconnect event

Steps to implement status callback are listed as follow:

1. Register to the device
2. Setup appropriate callback function (**`KsetVideoLossCallback2(),`** **`KsetVideoRecoveryCallback2(), KSetNetworkLossCallback()`**)
3. Event handling in the status callback function

```
   //---- prepare status callback here
   // Video lost
   void VideoLossCallBack(DWORD dwCallbackID , unsigned char VideoLossFlag)
   {
   // To Do: Add your video loss handle code here.
   }

   // Video recovery
   Void VideoRecoveryCallBack(DWORD dwCallbackID, unsigned char
VideoRecoveryFlag)
   {
   // To Do: Add your video recovery handle code here.
   }

   // Disconnect
   void NetworkLossCallback(DWORD dwCallbackID)
   {
   // To Do: Add your network loss handle code here.
   }



   //---- register to the server
   // Set video loss call back
   KsetVideoLossCallback2( hK, dwCallbackID, VideoLossCallBack);
```

```
// Set video recovery call back
 KsetVideoRecoveryCallback2( hK, dwCallbackID, VideoRecoveryCallBack);

// Set network loss (disconnect) call back
 KSetNetworkLossCallback(hK, dwCallbackID, NetworkLossCallback);
```

5 PTZ Integration

PTZ Integration Architecture

This material covers how to integrate PTZ protocol with prepared information.

In the product architecture, the PTZ operation is defined as transparent tunnel; in this way, the PTZ protocol information does not keep in the firmware, and user's application has to parse and prepare PTZ commands in the application side.

To shorten the integration process, SDK provides implemented and tested PTZ protocol files, so that application may just utilize the PTZ protocols that has been prepared.

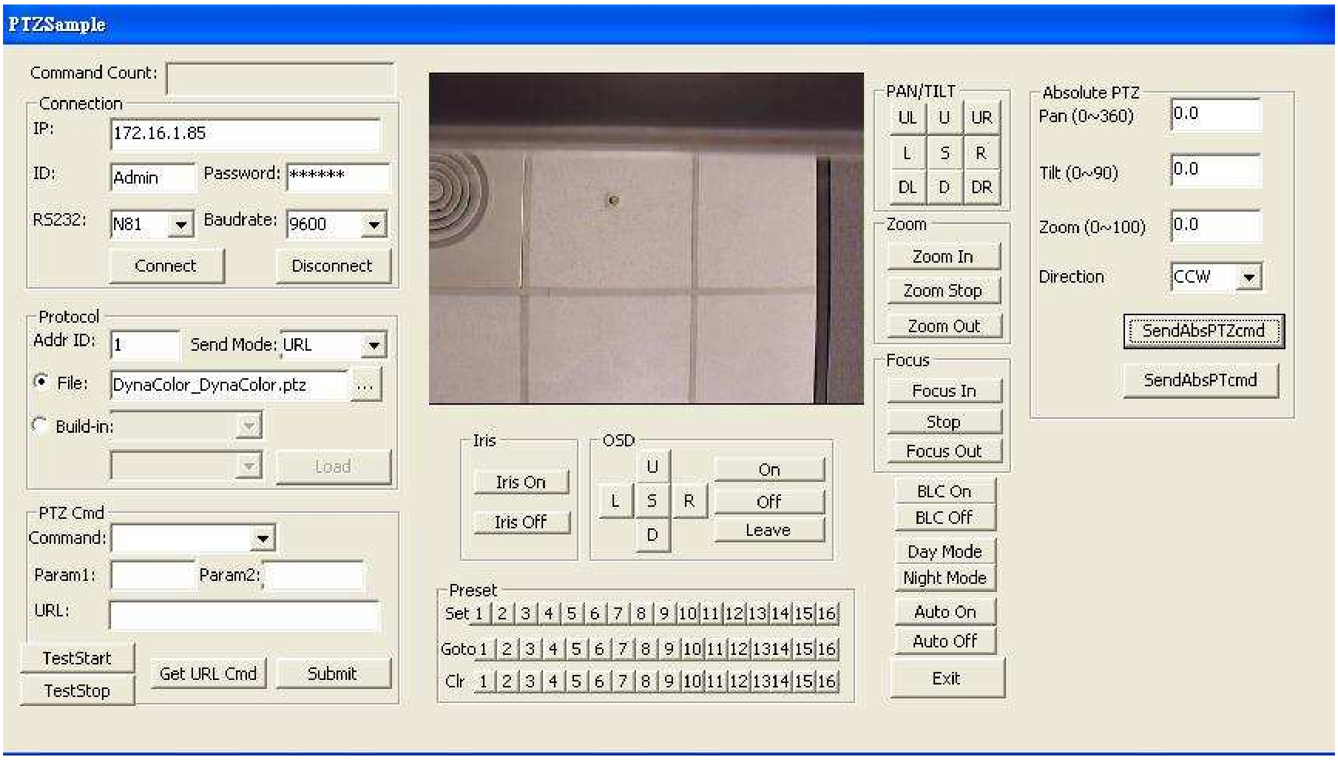
> **NOTE:** Firmware does not contain PTZ protocol information. User's application has to prepare the PTZ command string and execute the string directly

The benefits of the PTZ Integration architecture are listed as follow:

- Utilize tested protocols
- Provides PTZ operation command strings
- Provides important commands like Day and Night switch, Patrol, Pattern, IR, etc
- Provides OSD operation

PTZ Parser Source Code

Please refer to **`${SDK-DIR}\SDK\PTZSample`** for sample source code. Also, provides integrated PTZ protocol files under **`${SDK-DIR}\PTZ-Protocol`**.



PTZSample

Command Count:

Connection
IP: 172.16.1.85
ID: Admin
Password: *******
RS232: N81
Baudrate: 9600

Protocol
Addr ID: 1
Send Mode: URL
● File: DynaColor_DynaColor.ptz
○ Build-in:

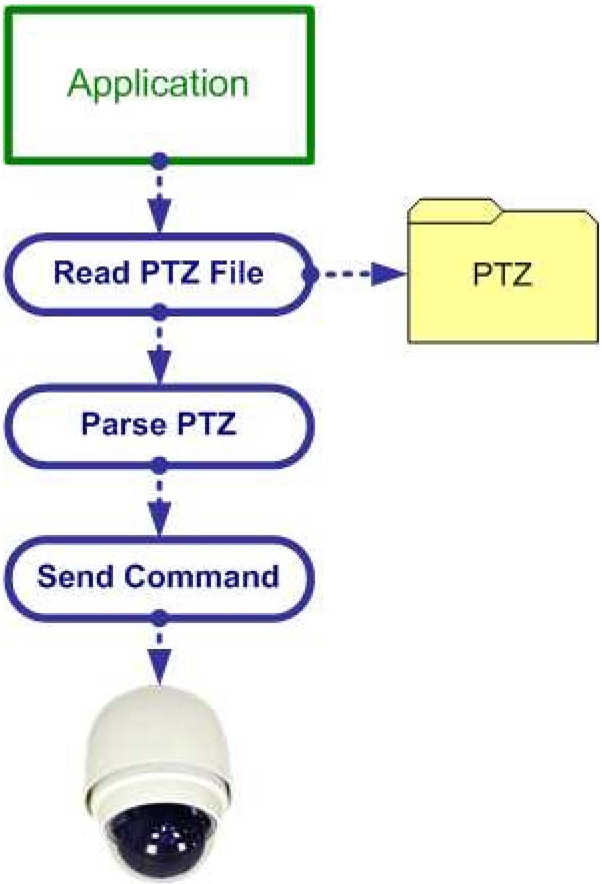
PTZ Cmd
Command:
Param1:
Param2:
URL:

Absolute PTZ
Pan (0~360) 0.0
Tilt (0~90) 0.0
Zoom (0~100) 0.0
Direction CCW

Steps to integrate a PTZ protocol include:

1. **Read PTZ File**: read PTZ protocol file specified
2. **Parse PTZ command**: parse the PTZ command rules, calculate the checksum and prepare the PTZ command
3. **Send Command**: sends PTZ command out with URL command or `netSend2ServerSerialPort()` function

(Most of new PTZ APIs in SDK 10000 V1.2 proceed step 1 and 2 at the same time)



PTZ Protocol Files ${SDK-DIR}\PTZ-Protocol

This section describes the definition of PTZ protocol files. Please get these files from **`${SDK-DIR}\PTZ-Protocol\`** directory. A sample fragment of the protocol file looks like follow

```
ADDRIDSTART; 1; 0;;;;
ADDRIDPOS; 2; 0;;;;
CHECKSUM; $B7=$B2+$B3+$B4+$B5+$B6;;;
INTERVAL;0;0;;;;
PANTILT;-5;-5;0xFF,0x01,0x00,0x14,0x3F,0x3F,0x93;;;
OSDON;0;0;0xFF,0x01,0x00,0x03,0x00,0x5F,0x63;;;
OSDUP;0;0;0xFF,0x01,0x00,0x08,0x00,0x0C,0x15;;;
OSDENTER;0;0;0xFF,0x01,0x02,0x00,0x00,0x00,0x03;;;
```

The protocol file contains following commands:

1. **`ADDRIDSTART`**: indicates the starting number of the address ID. Take above sample as an example (`ADDRIDSTART; 1; 0;;;;`), if the application is set to address ID as 3, then it starts at 1, so the calculated address ID is 3 (`0x03`);

2. **`ADDRIDPOS`**: indicates the position to replace with calculated address ID. Take above sample as an example (`ADDRIDPOS; 2; 0;;;;`), the address ID is at 2nd position of the command string. So, `PANTILT; -5, -5` command (`PANTILT;-5;-5;0xFF,0x01,0x00,0x14,0x3F,0x3F,0x93;;;`) will be replace as (`PANTILT;-5;-5;0xFF,0x03,0x00,0x14,0x3F,0x3F,0x93;;;`)

3. **`CHECKSUM`**: indicates the checksum rule, **+** is to run **AND** operation, **|** is to run **OR** operation, **^** is to run **XOR** operation. Take above sample as an example (`CHECKSUM; $B7=$B2+$B3+$B4+$B5+$B6;;;`), the checksum rule is to run **AND** operation for byte 2, byte 3, byte 4, byte 5 and byte 6, and the result is placed at byte 7. Then this becomes a final `PTZ command string`

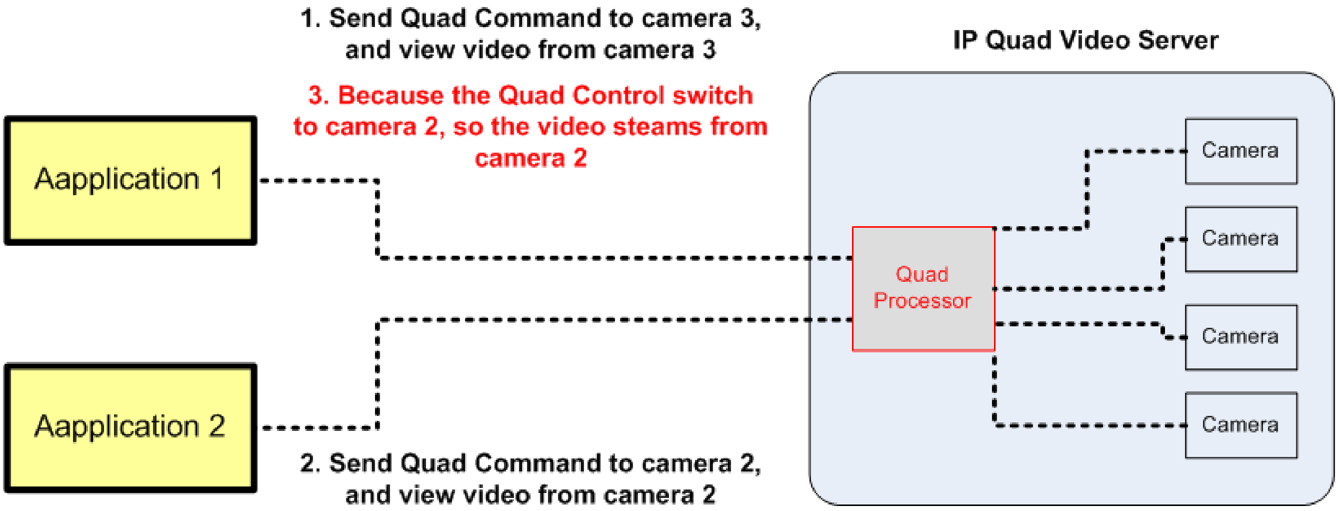
4. Application then sends the calculated `PTZ command string` out via normal serial port operation or URL command.

6 IP Quad Video Server Integration

IP Quad Architecture

IP Quad is a Quad processor which connects to 4 analog video sources then multiplexed by a quad processor; in this way, an IP Quad video server may generates 1 Full D1 video stream or 4 CIF video streams at the same time

IP Quad video server firmware contains URL commands, so that application may simply sends out the URL command to control the behavior of it.


1. Send Quad Command to camera 3, and view video from camera 3
3. Because the Quad Control switch to camera 2, so the video steams from camera 2
Aapplication 1
Aapplication 2
2. Send Quad Command to camera 2, and view video from camera 2
IP Quad Video Server
Quad Processor
Camera
Camera
Camera
Camera

NOTE: There is only one quad processor in the device, so when an application sends a URL command to the IP Quad video server, then the quad processor will execute the commands specified, and all connected application will receive the same result from quad processor.

IP Quad URL Commands

Application may just use URL Command to perform these tasks to setup and control Quad Video Server; for information that needs to retrieve from Quad Video Server (e.g. Retrieve video stream, record to files, motion detection event, digital input event), the calling methods are all the same as SDK-2000 v1.0.

IP Quad's quad control is based on URL Command, which means that you need to send out the URL Command to IP Quad to set certain parameters.

HTTP Code Status

HTTP Code	HTTP Text	Description
200	OK	The request has succeeded, but an application error can still occur, which will be returned as an application error code.
204	No Content	The server has fulfilled the request, but there is no new information to send back.
400	Bad Request	The request had bad syntax or was inherently impossible to be satisfied.
401	Unauthorized	The request requires user authentication or the authorization has been refused.
404	Not Found	The server has not found anything matching the request.
409	Conflict	The request could not be completed due to a conflict with the current state of the resource.
500	Internal Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
503	Service Unavailable	The server is unable to handle the request due to temporary overload.

Example :

Return success http context

```
HTTP/1.0 200 OK\r\n
Content-Type: text/plain\n
 \n
```

Return failed http context

```
HTTP/1.0 200 OK\r\n
Content-Type: text/plain\n
 \n
ERROR: error description
```

How to set display mode

Syntax	`http://192.168.1.1/cgi-bin/quad?DISPLAY=n`

How to get display mode

Syntax	`http://192.168.1.1/cgi-bin/quad?DISPLAY`

<parameter>	**<values>**	**Description**
DISPAY	n: 0~4	0: quad display 1: display channel 1 2: display channel 2 3: display channel 3 4: display channel 4

How to set OSD enabled

Syntax	`http://192.168.1.1/cgi-bin/quad?OSD_ENABLED=0xnn`

How to get OSD enabled status

Syntax	`http://192.168.1.1/cgi-bin/quad?OSD_ENABLED`

<parameter>	**<values>**	**Description**
OSD_ENABLED	0xnn : hexadecimal	BIT0: 1:title name enabled BIT1: 1:video loss enabled BIT2: 1:motion detect enabled BIT3: 1:date time enabled BIT4: 1:DIO status enabled BIT5: Reserved BIT6: Reserved BIT7: Reserved

How to set motion detect enabled

Syntax	`http://192.168.1.1/cgi-bin/quad?MOTION_ENABLED=0xnn`

How to get motion enabled status

Syntax	`http://192.168.1.1/cgi-bin/quad?MOTION_ENABLED`

<parameter>	**<values>**	**Description**
MOTION_ENABLED	0xnn : hexadecimal	BIT0: 1:channel 1 motion detect enabled BIT1: 1:channel 2 motion detect enabled BIT2: 1:channel 3 motion detect enabled BIT3: 1:channel 4 motion detect enabled BIT4: Reserved BIT5: Reserved BIT6: Reserved BIT7: Reserved

How to set sensitive for motion detect

Syntax	`http://192.168.1.1/cgi-bin/quad?CHANNEL=n&SENSITIVE=m`

How to get sensitive setting

Syntax	`http://192.168.1.1/cgi-bin/quad?CHANNEL=n&SENSITIVE`

<parameter>	**<values>**	**Description**
CHANNEL	n: 1~4	channel number
SENSITIVE	m: 0~15	0: more sensitive . .. 8: middle sensitive . .. 15: less sensitive

7 Advanced Topics

Callback Functions

This section lists the callback functions and its explanation for references.

Category	Function	Description
Decode	`KSetImageCallback()`	Callback functions to receive RGB buffer.
Event	`KSetDICallback()`	DI event triggers
Event	`KsetMotionDetectionCallback2()`	Motion detection event triggers
MPEG-4	`KSetRawDataCallback()`	Streaming raw data including Video and Audio. All data are in TCP v2.0 format.
MPEG-4	`KSetTimeCodeCallback()`	Timecode is sent to this callback function every time a frame arrives
Preview	`KSetAfterRenderCallback()`	Callback functions are called every time a frame is drawn on the screen. This is useful when user wants to draw their own OSD, Text or video intelligence information overlay on the preview window
Preview	`KSetResolutionChangeCallback()`	Callback function is called when resolution is changed.
RS-232	`KSetRS232DataCallback()`	RS-232/RS-422/RS-485 data arrives
System	`KsetVideoLossCallback2()`	Video loss event triggers.
System	`KsetVideoRecoveryCallback2()`	Video recovery event triggers.
System	`KSetNetworkLossCallback()`	Network loss is sent if disconnect.

Deals with RAW file format

This section describes the ways to deal with ".raw" data format, which is a standard of products.

including:

- Raw file header
- Raw file footer
- Data payload. (Video and Audio)

Deal Raw File Header and Footer

Here is the sample of catching raw header and footer. The detail is described in marked section.

```
/**  \brief Header of Raw file.
*
*/
typedef struct _tagRawFileHeader
{
     DWORD dwFirstB2;     // '00' '00' '01' 'B2'
     DWORD dwStreamType;       // 11 : TCP-1.0; 22 TCP-2.0 ; 0x22
     DWORD dwVideoType;        // 11 : ISO 14496; 22 : ... ; 0x11
     DWORD dwAudioType;        // 00 : NONE; 11 : PCM; ... ; 0x22
     DWORD dwReserve1[3];
     DWORD dwBiteRate;    // Bite rate 0 - 18 (Check out the programming guild for details)
     DWORD dwFps;         // Fps :1 - 30
     DWORD dwResolution; // Resolution : same as B2 header (Check out the programming guild for details)
     DWORD tBeginTime;         // UTC sec
     LONG lTimeBias;      // minute ; +8:00 = -480 ; UTC = local time + bias
     LONG lDaylightBias;  // minute ; +60
     DWORD dwReserve2[2]; // Reserve ; 0xFF
     DWORD dwExtendSize;       // 0x00
} stRawFileHeader;

/**  \brief Tail of Raw file.
*
*/
typedef struct _tagRawFileTail
{
     DWORD dwLastB2;           /**< '00' '00' '01' 'B2' */
     DWORD dwHeader;           /**< Must be 0xAC710517 (AC710517) */
     DWORD dwVersion;               /**< Must be 0x01000001 (1.0.0.1) */
     DWORD dwBeginTime;
     DWORD dwEndTime;
     DWORD dwGOP;
     DWORD dwGOPSize;
     DWORD dwFPS;
     DWORD dwWidth;
```

```
    DWORD dwHeight;
    DWORD dwFrameCount;
    DWORD dwTimeZone;
    char nTimeComplementarity;
    BYTE Reserve[7];
}stRawFileTail;
```

Get the Header and Footer

```
    // File open for read.
    if((m_fp = fopen(m_MediaConfig.PlayFileName, "rb")) == NULL)
    {
        return false;
    }

    // Set file size
    SetFileSize();

    // Read File header
    if(!ReadData(&m_RawFileHeader, sizeof(stRawFileHeader)))
    {
        return false;
    }

    // Read File Tail
    fseek(m_fp, m_dwFileSize - sizeof(RawFileTail_t), SEEK_SET);

    if(!ReadData(&m_RawFileTail, sizeof(stRawFileTail)))
    {
        return false;
    }

    // Set file position to first data, the first I frame.

    fseek(m_fp, sizeof(RawFileHeader_t), SEEK_SET);
```

Raw File Payload

The raw data format (video and audio) is described as follow:

Video Data: **I-Frame Data Structure** , **P-Frame Data Structure**, **Motion JPEG frame**, **H.264 frame**

Audio Data: **Audio frame**

Deals with Media Stream

This section describes the ways to deal with media stream, including:

- The raw data callback (Video and Audio)
- How to detect I-Frame
- Decode I-Frame only

Raw Data Format in TCP 2.0

MPEG-4 stream raw data format (video and audio) is described as follow:

Video Data: **I-Frame Data Structure** , **P-Frame Data Structure** , **Motion JPEG frame, H.264 frame**

Audio Data: **Audio frame**

Get Streaming Raw Data (Video + Audio)

Steps to get streaming raw data include:

1. Register to the IP devices
2. Setup kmpeg4 callback function

```
//---- prepare callback function when MPEG-4 raw data arrives

//7 types are needed
// 1. mpeg4
// 2. Audio PCM (not always with time stamp)
// 3. Audio PCM must with time stamp
// 4. MJPEG
// 5. H.264
// 6. Audio G711 mu law
// 7. Audio G711 a law
enum Raw_Data_Type
{
 EXCEPTION = 0,
 MPEG4_DATA = 1,
 AUDIO_PCM_DATA = 2,
 AUDIO_PCM_TIMESTAMP_DATA = 3,
 MJPEG_DATA = 4,
 H264_DATA = 5,
 AUDIO_G711A_DATA = 6,
 AUDIO_G711U_DATA = 7
};
void RawDataCallBack(DWORD id, DWORD dwDataType, BYTE* buf, DWORD len )
{
    switch (dwDataType)
{
      Case MPEG4_DATA:
//do something for video stream
      break;

      Case AUDIO_PCM_DATA:
//do something for audio stream
      break:

      Case AUDIO_PCM_TIMESTAMP_DATA:
//do something for audio stream
      break:

      Case MJPEG_DATA:
//do something for Motion JPEG stream
```

```
       break:

       Case H264_DATA:
//do something for H264 stream
       break:

       Case AUDIO_G711A_DATA:
//do something for audio stream
       break:

       Case AUDIO_G711U_DATA:
//do something for audio stream
       break:
     }

}
// Prepare your callback function first


//---- register server
HANDLE hK =  KOpenInterface();
// you should get HANDLE by KOpenInterface before Preview

// Set call back functions
 KSetRawDataCallback(hK, id, RawDataCallBack);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_UNICAST_PREVIEW;
 strcpy(mcc.UserID, “Your ID”);
 strcpy(mcc.Password, “Your Password”);
 mcc.RegisterPort = 6000;
 mcc.ControlPort = 6001;
 mcc.StreamingPort = 6002;
 mcc.MultiCastPort = 5000;
 mcc.HTTPPort = 80;
 strcpy(mcc.UniCastIP, “172.16.1.81”);
 strcpy(mcc.MultiCastIP, “225.5.6.81);

// Set media configuration file.
KSetMediaConfig2(hK, &mcc);
// Register
 KConnect(hK);
// Start Streaming
 KStartStreaming(hK);
```

```
// Start receiving data from KMpeg4
 KPlay(hK);



//---- below listing some steps if you need to terminate the whole process
 KStop(hK);
 KStopStreaming(hK);
 KDisconnect(hK);
 KCloseInterface(hK);
```

Detect I-Frame (key frame)

Steps to detect I-Frame in MPEG-4 raw data include:

1. Process in MPEG-4 raw data callback function
2. Check the MPEG-4 raw data format

Video data structure:

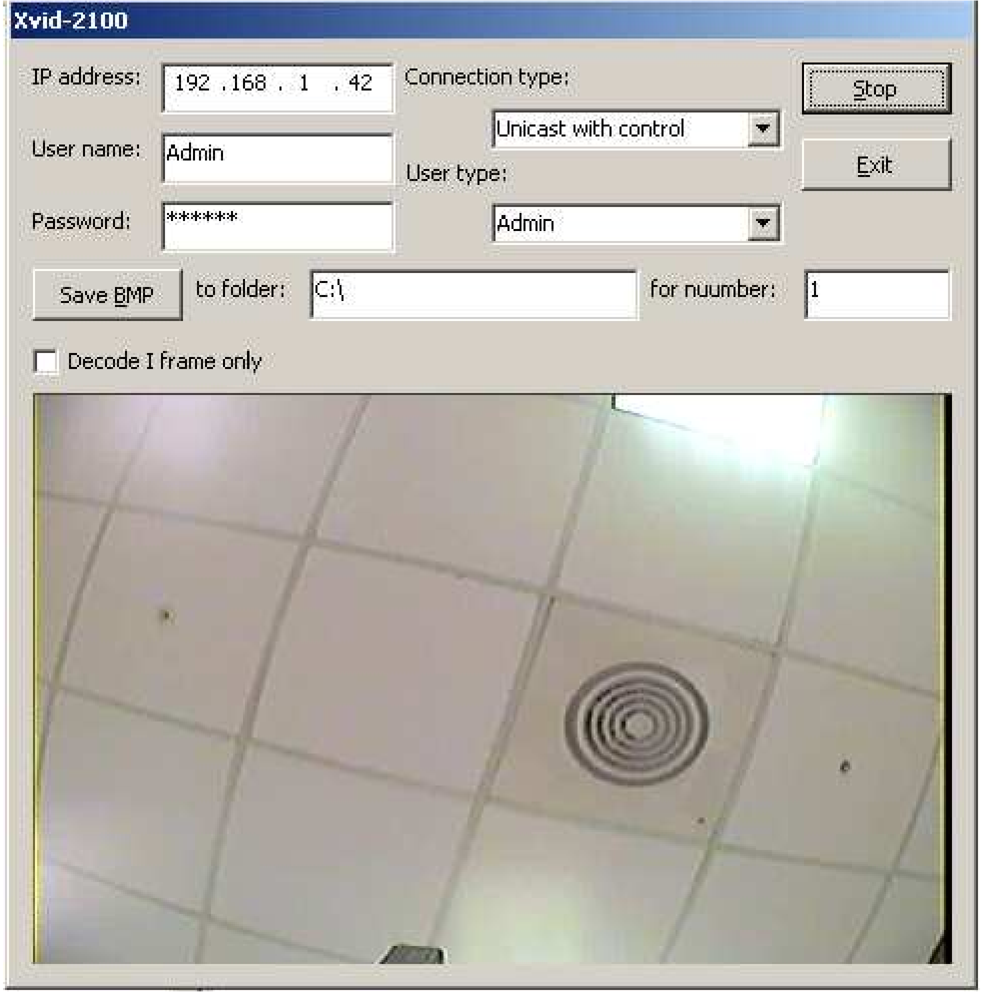
I Frame = **User Data** + **Bitstream Data** + **I-Frame Data**

```
// in C++ language example, here shows to know an I-Frame
// We suppose BYTE* buf is a continuous raw data for one frame
// compare 0xB3010000 with 4 bytes from the 75th byte in BYTE* buf
     DWORD f;
     CopyMemory( (BYTE*)&f, (buf+75), sizeof( DWORD ) );

     // an I-Frame
     if( f == 0xB3010000 )
  {
  }
  else ; //---- P-Frame
```

Decode MPEG-4 Stream with Xvid

MPEG-4 stream complies with standard ISO-14496-2 format and can be decoded with open source MPEG-4 software decoders, including FFMPEG, Xvid, DivX, etc.

Please refer to `${SDK-DIR}\SDK\Samples\DecodeSample` sample program.



Xvid-2100

IP address: 192 .168 . 1 . 42

User name: Admin

Password: ******

Connection type: Unicast with control

User type: Admin

to folder: C:\

for nuumber: 1

☐ Decode I frame only

Steps to use netSetMpeg4RawDataCallBack and decode by XVID:

1. Link libxvidcore.lib as Import Lib
2. Put xvidcore.dll in the same directory
3. Include xvid.h
4. Provide following initialize, create, decode, close xvid code.

```
   #include "xvid.h"

   DWORD m_vWidth;
   char  pOutBuf[720*576*3];

   xvid_dec_create_t   m_xvidDecHandle;
   xvid_gbl_init_t     xvid_gbl_init;
int                xvidret;
```

```
   //-----------------------------------------------------------------------
-
   // XVID Decord Init and Create ==>

          memset(&xvid_gbl_init, 0, sizeof(xvid_gbl_init));
       memset(&m_xvidDecHandle, 0, sizeof(m_xvidDecHandle));
          m_xvidDecHandle.version = XVID_VERSION;
       m_xvidDecHandle.height = 0;
       m_xvidDecHandle.width = 0;
       xvid_gbl_init.version = XVID_VERSION;
       xvidret = xvid_global(0, XVID_GBL_INIT, &xvid_gbl_init, 0);
          xvidret = xvid_decore(NULL, XVID_DEC_CREATE, &m_xvidDecHandle, NULL);

   //-----------------------------------------------------------------------
-
   // XVID Decord ==>  Put the code into the netSetMpeg4RawCallBack 's CallBack
Function

          xvidDecFrame.output.csp = XVID_CSP_BGR;
          xvidDecFrame.general = XVID_LOWDELAY|XVID_DEBLOCKY|XVID_DEBLOCKUV;
       xvidDecFrame.general = XVID_LOWDELAY;
       xvidDecFrame.version = XVID_VERSION;
       xvidDecFrame.output.plane[0] = pOutBuf;              // <<<<<<<

   //-----------------------------------------------------------------------
-
   // Output Buffer for the Decord out put

          // <<<<<<< The Video's Width Size => m_vWidth * 3, (a Pixel is 3 Bytes
(RGB))
          // <<<<<<< The m_vWidth can get from the Mpeg4 Raw Data
          // <<<<<<< (In the input buffer that first time the callback be called)
          // <<<<<<< Or can assign by yourself if you know what is the video's
width
          xvidDecFrame.output.stride[0] = m_vWidth * 3;

          xvidDecFrame.bitstream = pInBuf; // <<<<<<< The Mpeg4 Raw Data
       xvidDecFrame.length = Len;          // <<<<<<< Mpeg4 Raw Data's Length

       xvidret = xvid_decore(m_xvidDecHandle.handle, XVID_DEC_DECODE,
&xvidDecFrame, 0);
          // Todo : pOutBuf -> Display

   //-----------------------------------------------------------------------
-
```

```
// XVID Decord Close ==>

xvidret = xvid_decore(m_xvidDecHandle.handle, XVID_DEC_DESTROY, 0, 0);
```

Get RGB Image Data

Get RGB Image Data with Image Callback Function

Steps to get RGB image data with image callback function:

1. Register to the IP devices
2. Initialize stream
3. Start stream
4. Setup image callback function

```
   //---- prepare image callback function

   Void ImageCallBack(DWORD id, BYTE* pBuf, DWORD len, long w, long h)
   {
   // list sample below for save BMP file to “save.bmp” after get RGB data
     LPBITMAPINFO lpbih = (LPBITMAPINFO)pBuf ;
     Long lImageLen=(lpbih->bmiHeader).biSize \ (lpbih->bmiHeader).biSizeImage ;

     BITMAPFILEHEADER oHeader ;
     oHeader.bfType = 0x4d42 ;
     oHeader.bfReserved1 = 0;
     oHeader.bfReserved2 = 0;
     oHeader.bfSize = (DWORD)(sizeof(BITMAPFILEHEADER) \ +
(lpbih->bmiHeader).biSize + (lpbih->bmiHeader).biSizeImage) ;
    oHeader.bfOffBits = (DWORD)(sizeof(BITMAPFILEHEADER) +
(lpbih->bmiHeader).biSize) ;

    CFile oImage ;
    oImage.Open(“save.bmp”, CFile::modeCreate | CFile::modeWrite ) ;
    oImage.Write( &oHeader, sizeof(BITMAPFILEHEADER) );
    oImage.Write( pBuf, (lpbih->bmiHeader).biSize ) ;

    for(int i = lpbih->bmiHeader.biHeight-1 ; i >= 0 ; i--)
         oImage.Write( (pBuf+(lpbih->bmiHeader).biSize +
(i*lpbih->bmiHeader.biWidth*4)), lpbih->bmiHeader.biWidth*4) ;

    oImage.Close();
   }


   HANDLE hK =  KOpenInterface();
   // you should get HANDLE by KOpenInterface before Preview

   // Set call back functions
```

```
 KSetRawDataCallback(hK, id, RawDataCallBack);

// Set Display Informationm
 MEDIA_RENDER_INFO mri;
 mri.RenderInterface = DGDI;
 mri.hWnd = m_hWnd;             // Windows’ handle to draw
 mri.rec = m_rec;               // rec information.
 KSetRenderInfo(hK, &mri);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_UNICAST_PREVIEW;
 strcpy(mcc.UserID, “Your ID”);
 strcpy(mcc.Password, “Your Password”);
 mcc.RegisterPort = 6000;
 mcc.ControlPort = 6001;
 mcc.StreamingPort = 6002;
 mcc.MultiCastPort = 5000;
 mcc.HTTPPort = 80;
 strcpy(mcc.UniCastIP, “172.16.1.81”);
 strcpy(mcc.MultiCastIP, “225.5.6.81);

// Set media configuration file.
KsetMediaConfig2(hK, &mcc);
// Register
 KConnect(hK);
// Start Streaming
 KStartStreaming(hK);
// Start receiving data from KMpeg4
 KPlay(hK);

// Plug Image callback for get RGB Image
KSetImageCallback(hK, dwCallBackID, ImageCallBack);
.. .. ..
.. .. ..


// below listing some steps if you need to terminate the whole process
Stop(hK);
 KStopStreaming(hK);
 KDisconnect(hK);
 KCloseInterface(hK);
```

Save RGB Data into a BMP file

We can get raw data and save to other file format e.g. if we want to save the current frame to Bitmap file for website image index. Just as like as general computer file format the Bitmap file has itself format. The Bitmap file format has a **`BITMAPFILEHEADER`**, **`BITMAPINFORHEAR`** and bitmap bits. Luckily we just have to prepare the header because the bitmap bits that we can get from MPEG-4 Callback function. Below is the 24 bit Bitmap file example.

Steps to save RGB data into a BMP file include:

1. Register to the IP devices
2. Initialize stream
3. Start stream
4. Setup image callback function
5. Create **`BITMAPFILEHEADER`** data structure and write to file

NOTE: Please refer to **`/SDK/Samples/DecodeSample`** sample program for full source codes.

F

First we have to create the **`BITMAPFILEHEADER`** struct and write to file.

```
// Save 24bit BMP
long BufferSize = 720*480*3;
// Write out the file header
//
BITMAPFILEHEADER bfh;
memset( &bfh, 0, sizeof( bfh ) );
bfh.bfType = 'MB';
bfh.bfSize = sizeof( bfh ) + BufferSize + sizeof( BITMAPINFOHEADER );
bfh.bfOffBits = sizeof( BITMAPINFOHEADER ) + sizeof( BITMAPFILEHEADER );

DWORD Written = 0;
WriteFile( hf, &bfh, sizeof( bfh ), &Written, NULL );
```

Second we have to create the **`BITMAPINFOHEADER`** struct and write to file.

```
// Write the bitmap format
//
BITMAPINFOHEADER bih;
memset( &bih, 0, sizeof( bih ) );
bih.biSize = sizeof( bih );
bih.biWidth = 720;
bih.biHeight = -480;    //Save from down to up
bih.biPlanes = 1;
bih.biBitCount = 24;
Written = 0;
WriteFile( hf, &bih, sizeof( bih ), &Written, NULL );
```

Finally we only need to write the bitmap bits to file and close it.

```
// Write the bitmap bits
//
Written = 0;
WriteFile( hf, xvidDecFrame.output.plane[0], BufferSize, &Written, NULL );

// Close BMP file
CloseHandle( hf );
```

Save Recording to an AVI file

Steps to save recording data into an AVI file include:

1. Register to the IP devices
2. Sets MPEG-4 raw data callback
3. Sets FourCC type as “**vids**”
4. Sets FourCC handle as “**DX50**”
5. Calls AVI functions when receiving frames

NOTE: Please refer to **MSDN** sample or Microsoft web site for reference.

```
    A
VIFileInit(); // initializes the AVIFile library



    strcpy((char*)g_aviname, m_NormalSaveFile);  // file name
    g_aviframesize = (m_width * m_height * 3) / 2;

        // Is the file exist?
    FILE *fp = fopen(m_NormalSaveFile, "rb");  if (fp)  {
        fclose(fp);
        DeleteFile(m_NormalSaveFile); // delete it.
    }

    AVISTREAMINFO g_strhdr_out;
    BITMAPINFO g_header;
       // clear the struct
    memset(&g_strhdr_out, 0, sizeof(g_strhdr_out));

    g_strhdr_out.fccType             = mmioFOURCC('v', 'i', 'd', 's');// stream
type
    g_strhdr_out.fccHandler           = mmioFOURCC('D', 'X', '5', '0');
    g_strhdr_out.dwScale             = 1001;
    g_strhdr_out.dwRate              = (DWORD)(m_theFps * 1001);
    g_strhdr_out.dwSuggestedBufferSize  = g_aviframesize;

    g_header.biSize = 40;
    g_header.biWidth = m_width;
    g_header.biHeight = m_height;
    g_header.biPlanes = 1;
    g_header.biBitCount = 0;
    g_header.biCompression = g_strhdr_out.fccHandler;
    g_header.biSizeImage = g_aviframesize * 2;
    g_header.biXPelsPerMeter = 0;
    g_header.biYPelsPerMeter = 0;
    g_header.biClrUsed =0;
    g_header.biClrImportant =0;


        // Create a AVI file.
    hr = AVIFileOpen(&m_pAviFile, (char*)g_aviname, OF_WRITE | OF_CREATE, NULL);
    if (hr != AVIERR_OK) {
        AVIFileExit();
        return -1;
    }

    // Create a interface to the new stream.
    hr = AVIFileCreateStream(m_pAviFile, &m_pAviVideo, &g_strhdr_out);
    if (hr != AVIERR_OK) {
        AVIFileExit();
        return -1;
    }
        // sets the format of a stream at the specified position
    hr = AVIStreamSetFormat(m_pAviVideo, 0, &g_header, sizeof(g_header));
    if (hr != AVIERR_OK) {
        AVIFileExit();
        return -1;
    }

    m_AviFrameNo = 0;

    if (IFrame)
        m_AviFlag = AVIIF_KEYFRAME; // I frame
    else
        m_AviFlag = 0;

    // write data to stream
    hr = AVIStreamWrite(m_pAviVideo, m_AviFrameNo++, 1,
        (LPBYTE) (m_PreSaveFrame[j]),
        m_PreSaveFrameLen[j], m_AviFlag,
        NULL, NULL);

    if (hr != AVIERR_OK) {
        return -1; // Record AVIStreamWrite Error6
    }

        // Release the Stream
    AVIStreamRelease(m_pAviVideo);

        // Release the file
```

```
AVIFileRelease(m_pAviFile);

   // Release the AVIFile Libary
AVIFileExit();
```

Save Recording to an AVI file with SDK Function

Steps to save recording data into a AVI file include:

1. Connect to the IP devices
2. Sets File Writer Type to AVI
3. Start record

```
// Create a interface to the new stream.
HANDLE h = KopenInterface();
KsetMediaConfig2( h , cfg );
...
KConnect( h );
KStartStreaming( h );
...

KSetFileWriterType( h, FAVI /* 1 */ );   // To record by AVI mode
                                        // FAVI is a defined macro
KStartRecord( h, “FileName.avi”);
```

Register Control Connection Only

Register to control connection only if you only want to receive events from video server but not video data (for example: motion, DI). You can also send commands through control connection(for example: PTZ command, set motion…etc).

Steps to register with control connection only:

1. Call `KOpenInterface()` to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Call `KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)` to set connect config.
4. Set Contact type to `CONTACT_TYPE_CONTROL_ONLY`.
5. Call `KConnect(HANDLE).`
6. Call `KStartStreaming(HANDLE)` to get ready to receive.
7. Call `KPlay(HANDLE)` to start receive.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_CONTROL_ONLY;
 …
KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);

// Start Receive
 KPlay(hk);
```

Display text on screen

Steps to display text on screen while previewing.

1. Call **`KOpenInterface()`** to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
4. Set Contact type .
5. Call **`KConnect(HANDLE).`**
6. Call **`KStartStreaming(HANDLE)`** to get ready to receive.
7. Call **`KPlay(HANDLE)`** to start receive.
8. Call **`KSetTextOut()`** to diaply text.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_UNICAST_PREVIEW;
 …
KsetMediaConfig2(hK, &mcc);
// Set render info
 MEDIA_RENDER_INFO mri;
 KSetRenderInfo(h, &mri);

 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);

// Start Receive
 KPlay(hk);

// Display text
 KSetTextOut(h, 0, 0, 0, “123456789\0”, 9, true, false, false, “Arial”, 100,
 RGB(255, 255, 0), 2, RGB(0, 0, 255));
```

Use IPP codec

Steps to use IPP codec while previewing.

1. Call **`KOpenInterface()`** to get KMpeg4 handle.
2. Prepare IP address, port number, account, password, contact type..
3. Select codec by **`KSetCODECType(HANDLE h, int nType, int nChannel);`**
4. Call **`KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)`** to set connect config.
5. Set Contact type .
6. Call **`KConnect(HANDLE).`**
7. Call **`KStartStreaming(HANDLE)`** to get ready to receive.
8. Call **`KPlay(HANDLE)`** to start receive.

```
// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Select codec
 KSetCODECType(h, IPPCODEC, 0);
// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 mcc.ContactType = CONTACT_TYPE_UNICAST_PREVIEW;
 …
KsetMediaConfig2(hK, &mcc);


// Set render info
 MEDIA_RENDER_INFO mri;
 KSetRenderInfo(h, &mri);

 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);

// Start Receive
 KPlay(hk);
```

8 MPEG-4 Data Structure

Connection Type

Unicast Video and Control Connection

The section describes the mechanism on how to search IP surveillance products on network. With this mechanism, you can locate the devices on the network, then use URL commands to operate or manage those devices.

The function sends out a broadcast message, devices respond with detailed information, application then parse the replied information and parse the content with **NET_SEARCHSERVER** data structure.

Multicast Video + Control connection

The section describes the mechanism on how to search IP surveillance products on network. With this mechanism, you can locate the devices on the network, then use URL commands to operate or manage those devices.

Multicast Video(Without Connection)

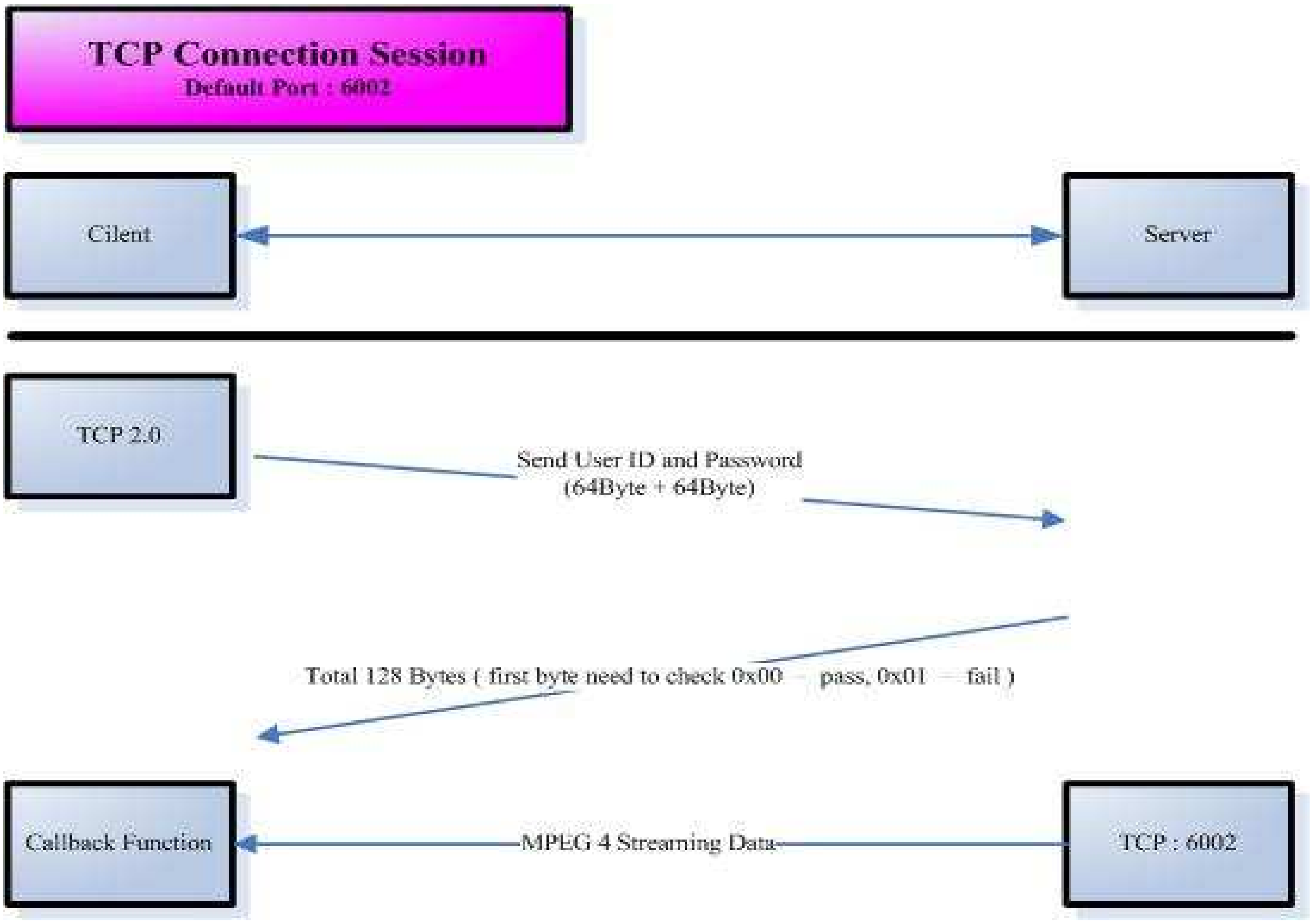
The section describes the mechanism on how to search IP surveillance products on network. With this mechanism, you can locate the devices on the network, then use URL commands to operate or manage those devices.

Unicast Video and Control

Connect to Video Server

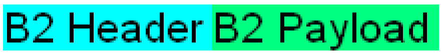
Here lists steps to build up the connection of getting audio/video streaming data.

1. Create a TCP socket connection that is needed to specific the IP and port. The default port is 6002.
2. Send a 128bytes command to video server. That includes User ID 64 bytes and Password 64 bytes.
3. Then we will get the response code. It are total 128 bytes code and includes a byte connect result.
4. Receive the data that will be audio/video streaming data.



Definition of B2 Frame

The B2 Frame is composed of B2 Header and B2 Payload. The length of B2 Header is fixed to 12 bytes. The length of B2 payload is variable length depends on the B2 MsgType defined in the B2 Header.



B2 Header	B2 Payload

There are two kinds of B2 Frame for video and audio usage.

Video B2 Frame (44 Byte):

```
typedef struct {
 B2_HEADER     header;
 PRIVATE_DATA   prdata;
} VIDEO_B2_FRAME;
```

Audio B2 Frame (28 Byte):

```
typedef struct {
 B2_HEADER header;
 struct timeval timestamp;
 unsigned char reserved[8];
} AUDIO_B2;
```

These structures will be detailed in rest of this chapter.

Mpeg4 Video Data Format

Video and Audio Frame

After the connection is established, this section introduces how to get the streaming data from video Server.

We use the private data header(0x000001B2) to be the header tag. When we receive the data tag is the 0x000001B2 and the follow is the struct B2_HEADER. If the msg_type of the B2_HEADER is 1 and this frame is the video frame. Another 2 is the audio frame.

Video frame

Mpeg4 streaming data has two kinds of video frame that is called I-Frame and P-Frame. There have some differenences. The I-Frame includes the sequence header that describes the information of decode(Bitstream data) like as below.

Header(0x000001B2) VIDEO_B2_FRAME	Bitstream Data	I-Frame data

The P-Frame is simpler than I-Frame. It doesn’t include the sequence header.

Header(0x000001B2) VIDEO_B2_FRAME	P-Frame data

I-Frame Data Structure

B2 Header for Video

```
typedef struct {
 B2_HEADER     header;
 PRIVATE_DATA   prdata;
} VIDEO_B2_FRAME;

#define DATA_TYPE_VEDIO_MPV4     1
#define B2_AUDIO_8KPCM           2
#define B2_AUDIO_8KPCM_TS        3
#define DATA_TYPE_VEDIO_JPEG     4
#define DATA_TYPE_VEDIO_H264     5
#define B2_AUDIO_G711A           6
#define B2_AUDIO_G711U           7
#define B2_AUDIO_G726_24         8
#define B2_AUDIO_G726_32         9
#define DATA_TYPE_AUDIO_PCM2     B2_AUDIO_8KPCM
#define DATA_TYPE_AUDIO_PCM3     B2_AUDIO_8KPCM_TS

typedef struct {
 unsigned char Key[4];  /* 00 00 01 B2 */
 unsigned char Type;
 unsigned char Stream_id;
 unsigned char Ext_b2_len;  //1: length of the ext.
                           //b2 private data appended to B2 Frame
 unsigned char Rsvd;
 unsigned int  Len;
} B2_HEADER;

typedef struct {
 unsigned long  Time;
 unsigned char  TimeZone;   /* mapping the TIME_ZONE in conf file. 0:-12,
                           ..., 24:+13 */
 unsigned char  VLoss;     /* 0: video loss, 1 : video ok */

 union{
      unsigned char Motion;
      struct {
           unsigned char reserved_bits1 :1;
           unsigned char MD1 :1;
           unsigned char MD2 :1;
           unsigned char MD3 :1;
           unsigned char reserved_bits2:3;
           unsigned char PIR :1;
```

```
    }MDb;

    struct {
        unsigned char MD1 :1;
        unsigned char MD2 :1;
        unsigned char MD3 :1;
        unsigned char MD4 :1;
        unsigned char reserved_bits2:3;
        unsigned char PIR :1;
    }QMDb;
};

unsigned char  DIO;        /* for DIs, 0: DI triggered. 1: no triggered */
unsigned int   Cnt;
unsigned char  Resolution; /* mapping the VIDEO_RESOLUTION in cond
                           file. 0:N720x480, ... */
unsigned char  BitRate;    /* mapping the VIDEO_BITRATE in cond file.
                           0:28K, ... */
unsigned char  FpsMode;    /* mapping the VIDEO_FPS in cond
                           file. 0:MODE1(constant), 1:0:MODE2 */
unsigned char  FpsNum;     /* in constant FPS mode, it indicates the
                            video server's constant
                            FPS number, i.e. atoi(VIDEO_FPS_NUM).
                            in variable FPS mode, in indicates
                            the variable FPS number which
                            was requested by the TCP host.
                            If it is not in TCP, it indicates
                            the variable FPS number, i.e.
                            atoi(VIDEO_VARIABLE_FPS) */
struct timeval TS;

unsigned short MDActives[3];  /*the number of active micro-blocks in the motion
                               region.The MDACtives[0] represents the number
                               of active micro-blocks in the motion region
                               1.The micro-block is 16x16 pixels in
                               the video image.
                              If the motion was not triggered, the
                              corresponding MDActives has to be zero.
                              In WPL encoder and QUAD video server, these fields
                              are fixed to 0 because it does not support this
                              feature.*/
unsigned char FixTimeZone;  // Reserved for SDK only. The firmware will set this
                               byte to 0x00.

unsigned char isReset : 1;
unsigned char isPre : 1;    // Pre-Frame; Need to decode , but not need to  render;
```

```
    unsigned char PreCounts : 6;  // Pre-Frame counts. Valid if Pre-Frame is
                                     enable(1), max : 0x07 = 127.
  } PRIVATE_DATA;
```

Name	Size
B2_HEADER	12 bytes (0x000001B2)
PRIVATE_DATA_B2	32 bytes

The user data segment total bytes : 44 bytes.

Bitstream Data

Name	**Size**
`Header`	`8 bytes`
`Data`	`2 bytes`
`Sequence header`	`4 bytes (0x00000100)`
`Sequence data`	`Around 15 bytes`
	`Around 29 bytes`

The Bitstream data segment total bytes : around 29 bytes (Header + Data + Sequence header + Sequence data).

I-Frame Data

Name	**Size**
`Header`	`8 bytes`
`Data`	`3 bytes`
`Frame data`	`N bytes`
	`11 + N bytes`

The I-Frame data segment total bytes : 11 bytes + N bytes(Header + Data + I-Frame data).

P-Frame Data Structure

B2 Header for Video

```
typedef struct {
 B2_HEADER     header;
 PRIVATE_DATA   prdata;
} VIDEO_B2_FRAME;

#define DATA_TYPE_VEDIO_MPV4     1
#define B2_AUDIO_8KPCM           2
#define B2_AUDIO_8KPCM_TS        3
#define DATA_TYPE_VEDIO_JPEG     4
#define DATA_TYPE_VEDIO_H264     5
#define B2_AUDIO_G711A           6
#define B2_AUDIO_G711U           7
#define B2_AUDIO_G726_24         8
#define B2_AUDIO_G726_32         9
#define DATA_TYPE_AUDIO_PCM2     B2_AUDIO_8KPCM
#define DATA_TYPE_AUDIO_PCM3     B2_AUDIO_8KPCM_TS

typedef struct {
 unsigned char Key[4];  /* 00 00 01 B2 */
 unsigned char Type;
 unsigned char Stream_id;
 unsigned char Ext_b2_len;  //1: length of the ext.
                            //b2 private data appended to B2 Frame
 unsigned char Rsvd;
 unsigned int  Len;
} B2_HEADER;

typedef struct {
 unsigned long  Time;
 unsigned char  TimeZone;   /* mapping the TIME_ZONE in conf file. 0:-12,
                            ..., 24:+13 */
 unsigned char  VLoss;     /* 0: video loss, 1 : video ok */

 union{
      unsigned char Motion;
      struct {
           unsigned char reserved_bits1 :1;
           unsigned char MD1 :1;
           unsigned char MD2 :1;
           unsigned char MD3 :1;
           unsigned char reserved_bits2:3;
           unsigned char PIR :1;
```

```
    }MDb;

    struct {
        unsigned char MD1 :1;
        unsigned char MD2 :1;
        unsigned char MD3 :1;
        unsigned char MD4 :1;
        unsigned char reserved_bits2:3;
        unsigned char PIR :1;
    }QMDb;
};

unsigned char  DIO;        /* for DIs, 0: DI triggered. 1: no triggered */
unsigned int   Cnt;
unsigned char  Resolution; /* mapping the VIDEO_RESOLUTION in cond
                           file. 0:N720x480, ... */
unsigned char  BitRate;    /* mapping the VIDEO_BITRATE in cond file.
                           0:28K, ... */
unsigned char  FpsMode;    /* mapping the VIDEO_FPS in cond
                           file. 0:MODE1(constant), 1:0:MODE2 */
unsigned char  FpsNum;     /* in constant FPS mode, it indicates the
                            video server's constant
                            FPS number, i.e. atoi(VIDEO_FPS_NUM).
                            in variable FPS mode, in indicates
                            the variable FPS number which
                            was requested by the TCP host.
                            If it is not in TCP, it indicates
                            the variable FPS number, i.e.
                            atoi(VIDEO_VARIABLE_FPS) */
struct timeval TS;

unsigned short MDActives[3];  /*the number of active micro-blocks in the motion
                               region.The MDACtives[0] represents the number
                               of active micro-blocks in the motion region
                               1.The micro-block is 16x16 pixels in
                               the video image.
                              If the motion was not triggered, the
                              corresponding MDActives has to be zero.
                              In WPL encoder and QUAD video server, these fields
                              are fixed to 0 because it does not support this
                              feature.*/
unsigned char FixTimeZone;  // Reserved for SDK only. The firmware will set this
                               byte to 0x00.

unsigned char isReset : 1;
unsigned char isPre : 1;    // Pre-Frame; Need to decode , but not need to  render;
```

```
  unsigned char PreCounts : 6;  // Pre-Frame counts. Valid if Pre-Frame is
                                   enable(1), max : 0x07 = 127.
} PRIVATE_DATA;
```

Name	Size
B2_HEADER	12 bytes (0x000001B2)
PRIVATE_DATA_B2	32 bytes

The user data segment total bytes : 44 bytes.

P-Frame Data

Name	**Size**
`Header`	`4 bytes`
`Frame data`	`N bytes`
	`4 + N bytes`

The P-Frame data segment total bytes : 4 bytes + N bytes(Header data + P-Frame data).

Code Mapping in B2 Header

1.Time Zone

Time Zone	time_zone in PRIVATE_DATA_NEW
-12	0
-11	1
-10	2
-09	3
-08	4
-07	5
-06	6
-05	7
-04	8
-03	9
-02	10
-01	11
+00	12
+01	13
+02	14
+03	15
+04	16
+05	17
+06	18
+07	19
+08	20
+09	21
+10	22
+11	23
+12	24
+13	25
other time zone setting 1	26
another time zone setting 2	27
…	…

2.Resolution

Video Resolution	resolution in PRIVATE_DATA_NEW	
	Binary Value	Hex Value
NTSC		
XNTSC720x480	00000000	0x00
XNTSC352x240	00000001	0x01
XNTSC160x112	00000010	0x02
XNTSC176x120	00000110	0x06
XNTSC640x480	01000000	0x40
XNTSC1280x720	01000001	0x41
XNTSC1280x960	01000010	0x42
XNTSC1280x1024	01000011	0x43
XNTSC1600x1200	01000100	0x44
XNTSC1920x1080	01000101	0x45
XNTSC320x240	01000110	0x46
XNTSC160x120	01000111	0x47
XNTSC2032x1920	01001000	0x48
XNTSC2592x1944	01001011	0x4B
XNTSC2048x1536	01001100	0x4C
PAL		
XPAL720x576	00000011	0x03
XPAL352x288	00000100	0x04
XPAL176x144	00000101	0x05
XPAL640x480	11000000	0xC0

3.Bitrate

Video Bitrate	bitrate in PRIVATE_DATA_NEW
28K	0
56K	1
128K	2
256K	3
384K	4
500K	5
750K	6
1M	7
1.2M	8
1.5M	9
2M	10
2.5M	11
3M	12
3.5M	13
4M	14
4.5M	15
5M	16
5.5M	17
6M	18

Note: In MJPEG mode and Variable Bitrate mode, this bitrate setting in B2 is not valid. It will be fixed at the current encoder bitrate setting which is for constant bit rate mode with MPEG4 or H.264 encoding.

Audio frame

The data structure of audio frame is simpler than video frame. We can see as below.

`AUDIO_B2(0x000001B2 )`	`Audio Frame data (audio 8K pcm payload data)`

```
#define DATA_TYPE_VEDIO_MPV4     1
#define B2_AUDIO_8KPCM           2
#define B2_AUDIO_8KPCM_TS        3
#define DATA_TYPE_VEDIO_JPEG     4
#define DATA_TYPE_VEDIO_H264     5
#define B2_AUDIO_G711A           6
#define B2_AUDIO_G711U           7
#define B2_AUDIO_G726_24         8
#define B2_AUDIO_G726_32         9
#define DATA_TYPE_AUDIO_PCM2     B2_AUDIO_8KPCM
#define DATA_TYPE_AUDIO_PCM3     B2_AUDIO_8KPCM_TS

typedef struct {
 unsigned char Key[4];  /* 00 00 01 B2 */
 unsigned char Type;
 unsigned char Stream_id;
 unsigned char Ext_b2_len;  //1: length of the ext.
                            //b2 private data appended to B2 Frame
 unsigned char Rsvd;
 unsigned int  Len;
} B2_HEADER;
```

```
typedef struct {
 B2_HEADER Head;
 struct timeval TS; //timestamp
 unsigned char Rsvd[8];
} AUDIO_B2_FRAME;
```

Name	**Size**
AUDIO_B2	28bytes (0x000001B2)
Audio Frame Data	N bytes
	28 + N bytes

The audio total bytes : AUDIO_B2 + FrameData (28 bytes + N)

Notice: The old version firmware send B2_HEADER(12 bytes) instead AUDIO_B2 (28 bytes)

Control Connect Session

Besides the video session we can get some of control from the control connection session.

Send a 128bytes command to the IP device.

Build a connection

When we connect to a video server by control session, we follow steps below to connect with video server.

1. Create a TCP socket connection that is needed to specific the IP and port. The default port is 6001.
2. Send a 128bytes command to video server. That includes User ID 32 bytes ,Token command and reserved bytes.
3. Then we will get the response code. It are total 128 bytes code and includes connect result, error code and reserve bytes
4. Receive the data that will be control data.

TCP Authentication Request and Response Frame

```
TCP Authentication Request and Response Frame
/* ##### definitions in msg_type in B2_HEADER ##### */
#define HEAD_MSG_VARIABLE_FPS_REQ    0x20
#define HEAD_MSG_PAUSE_ON_REQ        0x21
#define HEAD_MSG_PAUSE_OFF_REQ   0x22

/* ##### definitions in server_reply in B2_HEADER ##### */
#define HEAD_HOST_REQUEST   0
#define HEAD_SERVER_REPLY   1

typedef struct {
 unsigned char b2h[4];  /* A C T i */
unsigned short msg_type; /* not used. reset to 0 */
unsigned char server_reply; /* not used, reset to 0 */
unsigned char stream_id; /* same definition as B2_HEADER */
 unsigned int  len;
} B2_HEADER;

/* ##### definitions in encryption_type in TCP_AUTHEN_REQ #### */
#define NAME_ENCODED_NONE        0
#define NAME_ENCODED_BASE64      1
typedef struct {
 char user_name[32];
 char rsvd[28];
 unsigned short stream_id;   /* same definition as B2_HEADER */
 unsigned short encryption_type;
 int  user_pwd[64];
} TCP_AUTHEN_REQ;
```

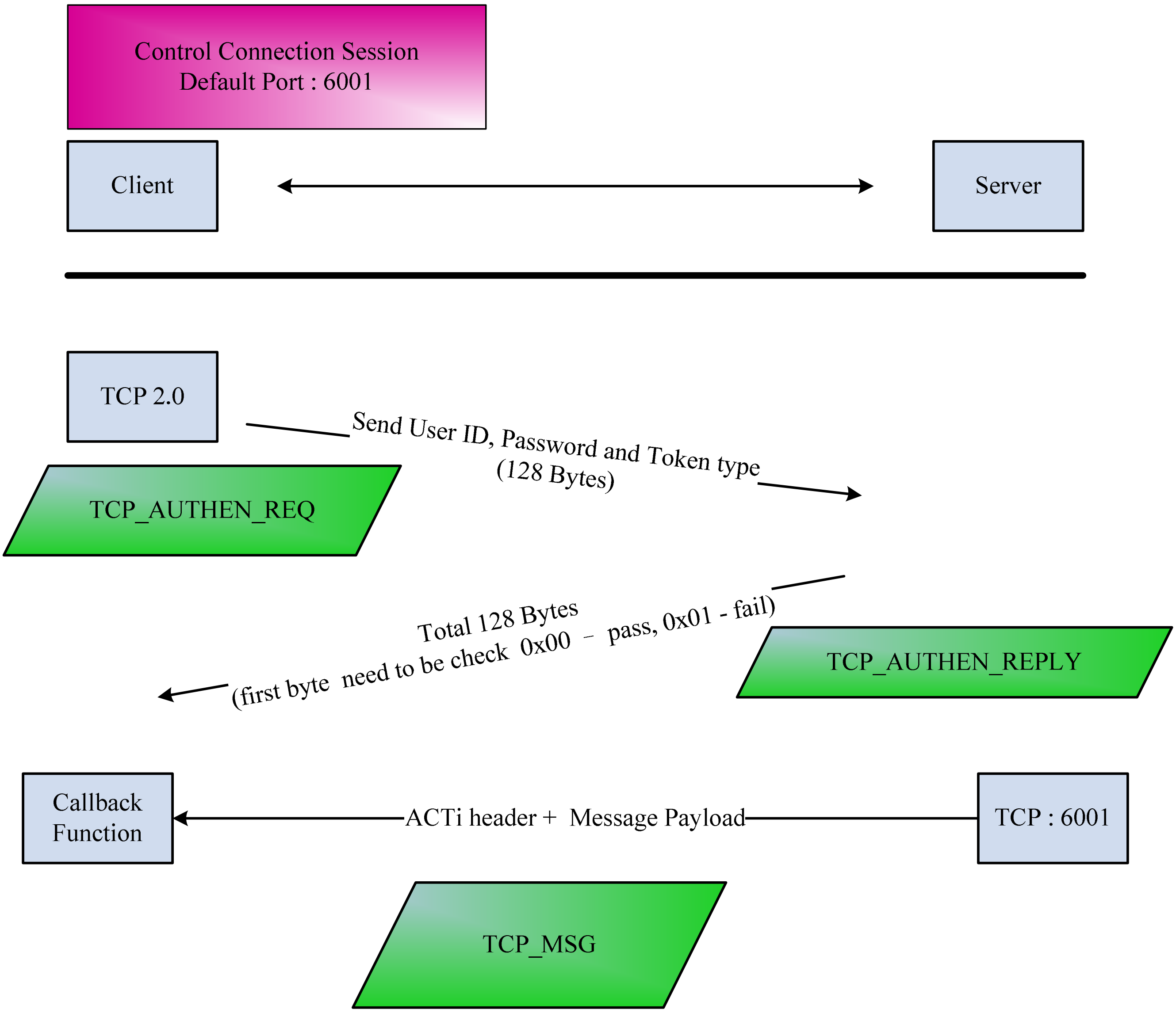
```
typedef struct {
 char status;
 char rsvd1;
 unsigned short stream_id;   /* same definition as B2_HEADER */
 int  sock;
 char camera_name[32];
 char rsvd2[88];
} TCP_AUTHEN_REPLY;

typedef struct {
 B2_HEADER header;
 unsigned char     msg[1];      /* variable length */
} TCP_MSG;
```

In msg_type = HEAD_MSG_VARIABLE_FPS_REQ, the msg[0] in TCP_MSG is the variable FPS number

In msg_type = HEAD_MSG_PAUSE_ON_REQ or HEAD_MSG_PAUSE_OFF_REQ, there is no msg[].

In the reply packet, the msg[0] is the return code. The definition of the return code is listed below.

```
#define TCP_REPLY_CODE_OK       0x00
#define TCP_REPLY_CODE_ERR  0x01
```


Control Connection Session
Default Port : 6001
Client
Server
TCP 2.0
Send User ID, Password and Token type
(128 Bytes)
TCP_AUTHEN_REQ
Total 128 Bytes
(first byte need to be check 0x00 – pass, 0x01 - fail)
TCP_AUTHEN_REPLY
Callback Function
ACTi header + Message Payload
TCP : 6001
TCP_MSG

Control Authentication Request and Response Frame

```
/* ##### definitions in msg_type in B2_HEADER */
/* LIVE CHECK used in the control session */
#define HEAD_MSG_LIVE       0x30
#define HEAD_MSG_EXIT       0x31
/* DIOs used in the control session */
#define HEAD_MSG_DIO_OUT    0x32
#define HEAD_MSG_DIO_STATUS  0x33
#define HEAD_MSG_DIO_INPUT    0x34   /* not used */
/* RS485 used in the control session */
#define HEAD_MSG_SERIAL_RECV  0x35   /* not used */
#define HEAD_MSG_SERIAL_SEND  0x36
/* AUDIO_IN used in the control session */
#define HEAD_MSG_AUDIO_PLAY     0x37
/* VIDEO LOSS used in the control session */
#define HEAD_MSG_VIDEO_LOSS   0x38   /* not used */
/* MOTION used in the control session */
#define HEAD_MSG_MOTION_DETECT 0x39 /* not used */
/* CAMERA NAME in the control session */
#define HEAD_MSG_CAMERA_NAME  0x40
/* ##### definitions in server_reply in B2_HEADER */
#define HEAD_HOST_REQUEST   0
#define HEAD_SERVER_REPLY   1

typedef struct {
 unsigned char b2h[4];  /* A C T i */
unsigned short msg_type; /* not used. reset to 0 */
unsigned char server_reply; /* not used, reset to 0 */
unsigned char stream_id; /* same definition as B2_HEADER */
 unsigned int  len;
} B2_HEADER;
```

```
typedef struct {
 char user[32];
 int  token;
 char reserved[24];
 unsigned short stream_id; /* same definition as B2_HEADER */
 unsigned short encryption_type; /* same definition as CP_AUTHEN_REQ */
 char pwd[64];
} CTRL_REQ;

/* ##### definitions in the result in CTRL_RSP ##### */
#define RSP_OK                  0x00
#define RSP_ERR                 0x01
/* ##### definitions in the err_code in CTRL_RSP ##### */
#define ERR_NO_ERROR            0x00000000
#define ERR_ACCOUNT             0x00010001
#define ERR_UNKNOWN_TOKEN       0x00010002
#define ERR_CTRL_TOKEN_BUSY     0x00010003
#define ERR_AUDIO_TOKEN_BUSY    0x00010004
#define ERR_AUDIO_NOT_SUPPORT   0x00010005

typedef struct {
 char result;
 char reserved1;
 unsigned short stream_id; /* same definition as B2_HEADER */
 char reserved1[3];
 int  err_code;
 int  ip_addr;
 char reserved2[116];
} CTRL_RSP;
```

```
typedef struct {
 B2_HEADER header;
 unsigned char  msg[1];    /* variable length */
} CTRL_MSG_FRAME;
```

DOs coding in the msg[0] (1byte):

Bit[4] : DO2, Bit[3] : DO1, Bit[1]=DI2, Bit[0]:DI1, where 1: high level of DO, 0: low level of DO.

RS485 coding in msg[] (variable length):

Data string of the RS485/RS422/RS232 data

Camera name coding in msg[] (max 31 bytes) :

Encoder's VIDEO_CAMERA_NAME setting

Audio data in msg[] (fixed to 4096 bytes):

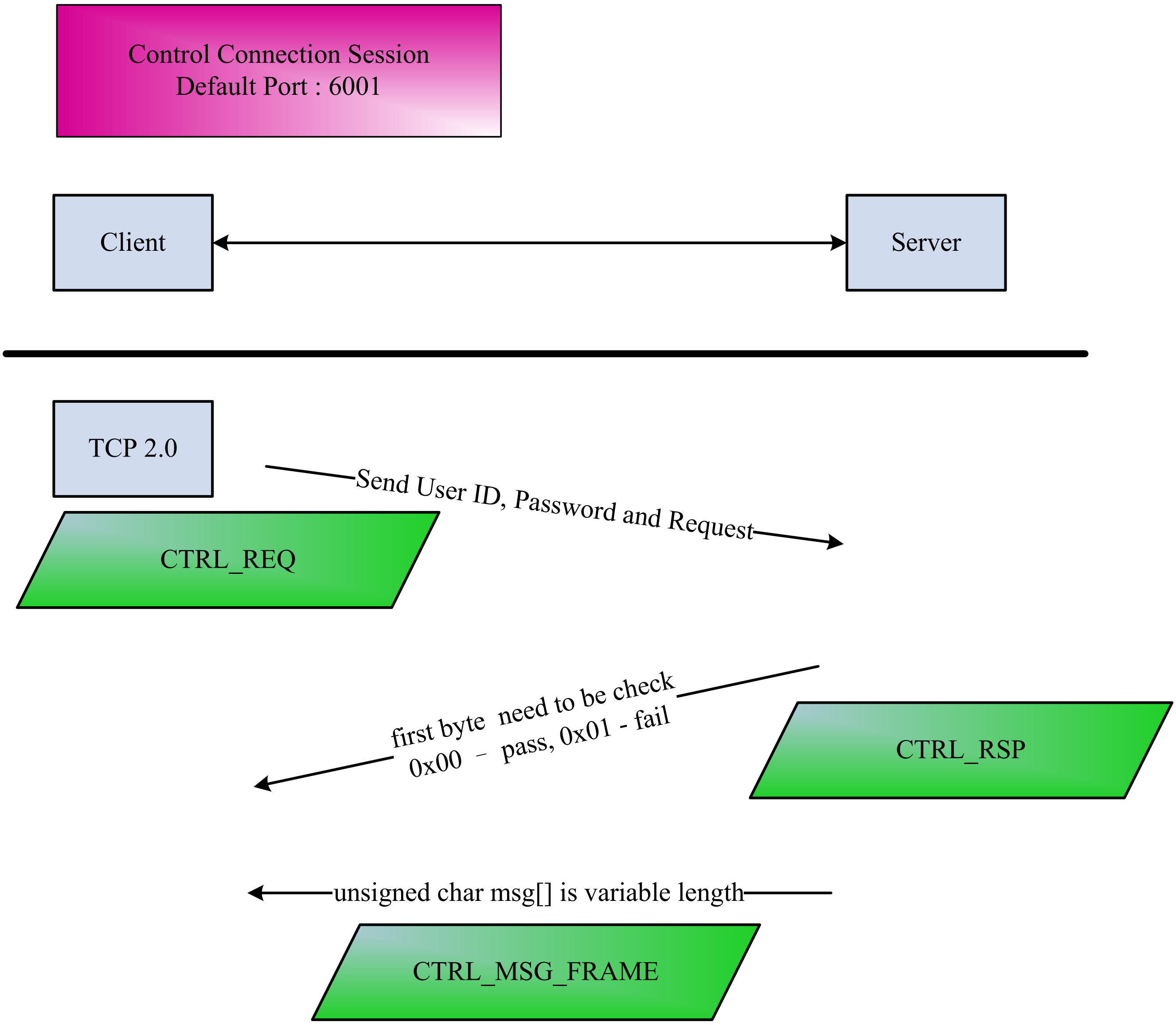
Audio data in host

Motion coding in the msg[0] (1byte):

Bit[1]: motion region 1, Bit[2]: motion region 2, Bit[3]: motion region 3, where 0: no motion, 1: detected motion

Video Loss coding in the msg[0] (1byte):

0: Video Loss, 1: Video Lock


Control Connection Session
Default Port : 6001
Client
Server
TCP 2.0
Send User ID, Password and Request
CTRL_REQ
first byte need to be check
0x00 – pass, 0x01 - fail
CTRL_RSP
unsigned char msg[] is variable length
CTRL_MSG_FRAME

9

JPEG-compressed Video Data Structure

JPEG-compressed Video Data Format

We won’t repeat most topics which described well in chap.8 . This chapter will concentrate on Motion JPEG data structure.

Motion JPEG and Audio Frame

After the connection is established, this section introduces how to get the streaming data from video Server.

We use the private data header(0x000001B2) to be the header tag. When we receive the data tag is the 0x000001B2 and the follow is the struct B2_HEADER. If the msg_type of the B2_HEADER is 4 and this frame is the Motion JPEG frame. Others 2 and 3 are the audio frame.

Motion JPEG frame In TCP Session

MJPEG streaming data is formed by JPEGs. The JPEG-Frame of TCP concept is described below. The JPEG header and JPEG data composite a complete JPEG picture, which can be easily decode by JPEG support library. (If you are looking for more information about B2 header, please refer to chap 8. **B2**)

Header(0x000001B2) VIDEO_B2_FRAME	JPEG Header	JPEG data

Motion JPEG frame In RTP Session

The JPEG-Frame of RTP concept is described below. JPEG header is modified in RTP session, we will describe later. (If you are looking for more information about RTP Header, please refer to RFC 1889)

RTP Header	RTP/JPEG Header	JPEG QUANT Header	JPEG data	Header(0x000001B2) VIDEO_B2_FRAME

TCP/MJPEG Header

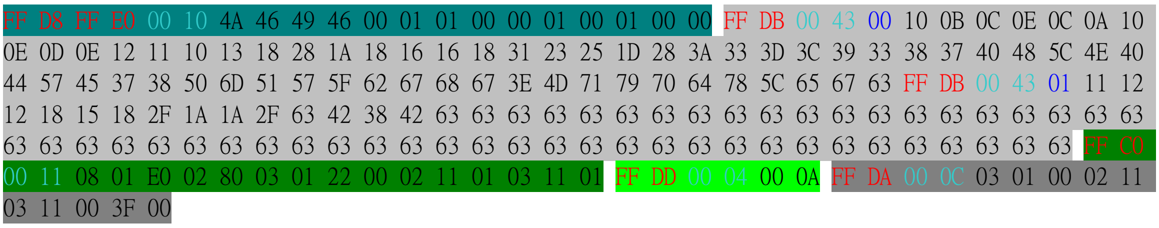
We section a part of description from “**JPEG File Interchange Format**” document to describe composition in TCP/JPEG header. The document can be acquired on **http://www.jpeg.org/** .

There are 5 parts in TCP/MJPEG header: SOI, Quant, SOF, DRI, and SOS.

Example of TCP/MJPEG Header

TCP MJPEG FRAME：

MJPEG ：



```
FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00 01 00 01 00 00 FF DB 00 43 00 10 0B 0C 0E 0C 0A 10
0E 0D 0E 12 11 10 13 18 28 1A 18 16 16 18 31 23 25 1D 28 3A 33 3D 3C 39 33 38 37 40 48 5C 4E 40
44 57 45 37 38 50 6D 51 57 5F 62 67 68 67 3E 4D 71 79 70 64 78 5C 65 67 63 FF DB 00 43 01 11 12
12 18 15 18 2F 1A 1A 2F 63 42 38 42 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 FF C0
00 11 08 01 E0 02 80 03 01 22 00 02 11 01 03 11 01 FF DD 00 04 00 0A FF DA 00 0C 03 01 00 02 11
03 11 00 3F 00
```

JPEG SOI：

FF	D8	FF	E0	JPEG SOI

length (2 bytes) Total APP0 field byte count, including the byte count value (2 bytes), but excluding the APP0 marker itself

identifier (5 bytes) = X'4A', X'46', X'49', X'46', X'00'
This zero terminated string (“JFIF”) uniquely identifies this APP0 marker. This string shall have zero parity (bit 7=0).

version (2 bytes) = X'0102'
The most significant byte is used for major revisions, the least significant byte for minor revisions. Version 1.02 is the current released revision.

units (1 byte) Units for the X and Y densities.
units = 0: no units, X and Y specify the pixel aspect ratio
units = 1: X and Y are dots per inch
units = 2: X and Y are dots per cm

Xdensity (2 bytes) Horizontal pixel density

Ydensity (2 bytes) Vertical pixel density
Xthumbnail (1 byte) Thumbnail horizontal pixel count
Ythumbnail (1 byte) Thumbnail vertical pixel count
(RGB)n (3n bytes) Packed (24-bit) RGB values for the thumbnail
pixels, n = Xthumbnail * Ythumbnail

JPEG QUANT：

FF	DB	2Bytes QUANT LEN	1Bytes JPEG QUANT ID	JPEG QUANT DATA

JPEG SOF：

FF	C0	JPEG SOF

length (2 bytes)
Precision (1 byte)
Height (2 bytes)
Width (2 bytes)
Type (1 byte) get the type, skip components, comp 0(1)
…

JPEG DRI：

FF	DD	JPEG DRI

…

JPEG SOS：

FF	DA	JPEG SOS

RTP/MJPEG Header

We section a part of description from “rfc2435-RTP” document to describe composition in RTP/MJPEG header.

Main header

Type-specific	Fragment Offset (24 bits)

Type (8 bits)	Q (8 bits)	Width (8 bits)	Height (8 bits)

1. Type-specific: 8 bits
Interpretation depends on the value of the type field. If no interpretation is specified, this field MUST be zeroed on transmission and ignored on reception.

2. Fragment Offset: 24 bits
The Fragment Offset is the offset in bytes of the current packet in the JPEG frame data. This value is encoded in network byte order (most significant byte first). The Fragment Offset plus the length of the payload data in the packet MUST NOT exceed 2^24 bytes.

3. Type: 8 bits
The type field specifies the information that would otherwise be present in a JPEG abbreviated table-specification as well as the additional JFIF-style parameters not defined by JPEG. Types 0-63 are reserved as fixed, well-known mappings to be defined by this document and future revisions of this document. Types 64-127 are the same as types 0-63, except that restart markers are present in the JPEG data and a Restart Marker header appears immediately following the main JPEG header. Types 128-255 are free to be dynamically defined by a session setup protocol.

4. Q: 8 bits

The Q field defines the quantization tables for this frame. Q values 0-127 indicate the quantization tables are computed using an algorithm determined by the Type field (see below). Q values 128-255 indicate that a Quantization Table header appears after the main JPEG header (and the Restart Marker header, if present) in the first packet of the frame (fragment offset 0). This header can be used to explicitly specify the quantization tables in-band.

5. Width: 8 bits
This field encodes the width of the image in 8-pixel multiples (e.g., a width of 40 denotes an image 320 pixels wide). The maximum width is 2040 pixels.

6. Height: 8 bits
This field encodes the height of the image in 8-pixel multiples (e.g., a height of 30 denotes an image 240 pixels tall). When encoding interlaced video, this is the height of a video field, since fields are individually JPEG encoded. The maximum height is 2040 pixels.

Restart Marker header

This header MUST be present immediately after the main JPEG header when using types 64-127.

Restart Interval(16 bits)	F (1 bit)	L (1 bit)	Restart Count (14 bits)

Quantization Table header

This header MUST be present after the main JPEG header (and after the Restart Marker header, if present) when using Q values 128-255. It provides a way to specify the quantization tables associated with this Q value in-band.

MBZ(8 bits)	Precision (8 bit)	Length (16 bit)	Quantization Table Data (Length bits)

Example of RTP/MJPEG Header

RTP Header：

```
/*
* RTP data header from RFC1889
*/
typedef struct {
unsigned int version:2; /* protocol version */
unsigned int p:1; /* padding flag */
unsigned int x:1; /* header extension flag */
unsigned int cc:4; /* CSRC count */
unsigned int m:1; /* marker bit */
unsigned int pt:7; /* payload type */
u_int16 seq; /* sequence number */
u_int32 ts; /* timestamp */
u_int32 ssrc; /* synchronization source */
u_int32 csrc[1]; /* optional CSRC list */
} rtp_hdr_t;
```

RTP MJPEG FRAME：

```
00 00 00 00 41 FF 50 3C
00 0A FF FF
00 00 00 80
1A 18 16 16 18 31 23 25 1D 28 3A 33 3D 3C 39 33 38 37 40 48 5C 4E 40 44 57 45 37 38 50 6D 51 57
5F 62 67 68 67 3E 4D 71 79 70 64 78 5C 65 67 63 11 12 12 18 15 18 2F 1A 1A 2F 63 42 38 42 63 63
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 C9 B5 B7 6B 89 B9 CE D1 5B 91 A0 8D 00 50 29 D1
```

RTP JPEG HEADER：

```
00 00 00 00 41 FF 50 3C
```

```
// Annotations in this section describe how rtp/jpeg header be generated from normal JPEG
// header (TCP/MJPEG header).
//
struct jpeghdr
{
     unsigned int tspec:8;       // 0 not used
     unsigned int off:24;        // 0 not used
     unsigned char type;         // JPEG SOF [11]
                                 // if SOF[11] == 0x21 Then type = 0;
                                 // else Then type = 1;
                                 // If jpeghdr_rst.dri Then type |= 0x40;
     unsigned char q;            // 0xff
     unsigned char width;        // JPEG SOF[5]; JPEG SOF[6];
                                 // (SOF[5]<<8 | SOF[6]) >>3
     unsigned char height;       // JPEG SOF[7]; JPEG SOF[8];
                                 // (SOF[7]<<8 | SOF[8]) >>3
};
```

RTP RESTART MARKER HEADER：

```
00 0A FF FF
struct jpeghdr_rst
{
     unsigned short dri;        // (JPEG DRI[4]<<8) | JPEG DRI[5]
     unsigned short f:1;        // always 1
     unsigned short l:1;        // always 1
     unsigned short count:14;   // always 0x3fff
};
```

RTP QUANTIZATION TABLE HEADER：

```
00 00 00 80
struct jpeghdr_qtable
{
     u_int8 mbz;           // not used
     u_int8 precision;     // JPEG SOF[4]
                           // if SOF[4] < 8 then precision = 0
                           // else then precision = (SOF[4]-8)/8
     u_int16 length;       // two bye after FF DB
};
```

RTP QUANTIZATION TABLE DATA：

The length of this table is “jpeghdr_qtable::length”; The table length of MJPEG is 64*2.

```
1A 18 16 16 18 31 23 25 1D 28 3A 33 3D 3C 39 33 38 37 40 48 5C 4E 40 44 57 45 37 38 50 6D 51 57
5F 62 67 68 67 3E 4D 71 79 70 64 78 5C 65 67 63 11 12 12 18 15 18 2F 1A 1A 2F 63 42 38 42 63 63
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 C9 B5 B7 6B 89 B9 CE D1 5B 91 A0 8D 00 50 29 D1
```

10

H.264 Video Data Structure

H -compressed Video Data Format

We won’t repeat most topics which described well in chap.8 . This chapter will concentrate on H.264 data structure.

H.264 and Audio Frame

After the connection is established, this section introduces how to get the streaming data from video Server.

We use the private data header(0x000001B2) to be the header tag. When we receive the data tag is the 0x000001B2 and the follow is the struct B2_HEADER. If the msg_type of the B2_HEADER is 5 and this frame is the H.264 frame. Others 2 and 3 are the audio frame.

H.264 frame In TCP Session

The H264-Frame of TCP concept is described below. (If you are looking for more information about B2 header, please refer to chap 8. **B2**)
If the H.264 Frame is an I Frame, there will be a H.264 Sequence Header behind VIDEO_B2_FRAME.

`Header(0x000001B2 )`
`VIDEO_B2_FRAME`
	`H.264 Frame data`

H.264 frame In C SDK 10000 connection

It’s easy to connect a device which has H.264 ability, just like we mentioned in previous chapters. We can examine the streaming from device by raw data callback.

Steps to register to device:

1. Prepare the callback function for raw data streaming.
2. Call `KOpenInterface()` to get KMpeg4 handle.
3. Set raw data callback by `KSetRawDataCallback()`
4. Prepare IP address, port number, account, password, contact type..
5. Call `KsetMediaConfig2(HANDLE, MEDIA_CONNECTION_CONFIG2)` to set connect config.
6. Call `KConnect(HANDLE).`

7. Call `KStartStreaming(HANDLE)` to get ready to receive.

```
//---- prepare callback function when MPEG-4 raw data arrives

//7 types are needed
// 1. mpeg4
// 2. Audio PCM (not always with time stamp)
// 3. Audio PCM must with time stamp
// 4. MJPEG
// 5. H.264
// 6. Audio G711 mu law
// 7. Audio G711 a law
enum Raw_Data_Type
{
 EXCEPTION = 0,
 MPEG4_DATA = 1,
 AUDIO_PCM_DATA = 2,
 AUDIO_PCM_TIMESTAMP_DATA = 3,
 MJPEG_DATA = 4,
 H264_DATA = 5,
 AUDIO_G711A_DATA = 6,
 AUDIO_G711U_DATA = 7
};
void RawDataCallBack(DWORD id, DWORD dwDataType, BYTE* buf, DWORD len )
{
    switch (dwDataType)
{
       Case MPEG4_DATA:
//do something for video stream
       break;

       Case AUDIO_PCM_DATA:
//do something for audio stream
       break:

       Case AUDIO_PCM_TIMESTAMP_DATA:
//do something for audio stream
       break:

       Case MJPEG_DATA:
//do something for Motion JPEG stream
       break:

       Case H264_DATA:
//do something for H264 stream
       break:
```

```
      Case AUDIO_G711A_DATA:
//do something for audio stream
      break:

      Case AUDIO_G711U_DATA:
//do something for audio stream
      break:
     }
}
// Prepare your callback function first

// you should get HANDLE by KOpenInterface before Preview
HANDLE hK = KOpenInterface();

// Set call back functions
 KSetRawDataCallback(hK, id, fnRawCallback);

// Prepare USER_INFO data structure by filling IP address, account, password.
 MEDIA_CONNECTION_CONFIG2 mcc;
// Set your connection information into struct mcc.
 …
KsetMediaConfig2(hK, &mcc);
 KConnect(hK);

// Start Streaming
 KStartStreaming(hK);
```

H.264 frame In RTP Session

The H.264-Frame of RTP concept is different from JPEG/RTP concept. The NAL (Network Abstraction Layer) unit in header indicates the type of the pocket. (If you are looking for more information about H.264/RTP, please refer to RFC 3984)

NAL		
1 bits	2 bits	5 bits
F	NRI	TYPE

The NAL unit

The semantics of the components of the NAL unit type octet, as specified in the H.264 specification, are described briefly below.

F: 1 bit
forbidden_zero_bit. The H.264 specification declares a value of 1 as a syntax violation.

NRI: 2 bits
nal_ref_idc. A value of 00 indicates that the content of the NAL unit is not used to reconstruct reference pictures for inter picture prediction. Such NAL units can be discarded without risking the integrity of the reference pictures. Values greater than 00 indicate that the decoding of the NAL unit is required to maintain the integrity of the reference pictures.

Type: 5 bits
nal_unit_type. This component specifies the NAL unit payload type as defined bellow.

Type	Packet	Type name
1-23	NAL unit	Single NAL unit packet per H.264
24	STAP-A	Single-time aggregation packet
25	STAP-B	Single-time aggregation packet
26	MTAP16	Multi-time aggregation packet
27	MTAP24	Multi-time aggregation packet
28	FU-A	Fragmentation unit
29	FU-B	Fragmentation unit

Summary of NAL unit types and their payload structures

In current devices, we use 2 types of pocket. One is the “Sequence Header ”, and the other is ”FU-A”(Fragmentation Units).

Sequence Header Pocket (type 1-23) :

The sequence header pocket should be looked like this.

RTP Header (It's detailed in previous chapter.)	NAL (1 byte)	H.264 Sequence Header (unprocessed)	Header(0x000001B2) VIDEO_B2_FRAME

When the “NAL” is 0x67 (type of NAL is 1 to 23), the received pocket is “Sequence Header Pocket”. We need further process to get complete “H.264 Sequence Header”.
We have to extend NAL first, then append “unprocessed H.264 Sequence Header“ later. The result will be “Complete H.264 Sequence Header”.

Here is the concept code in C++.

```
if( h261_nal->TYPE >= 1 && h261_nal->TYPE <=23 )
{
 unsigned char header[5];
 unsigned char szCompleteH264SequenceHeader[2048];
      // NAL
      if( psz_buf[RTP_HEADER_LEN] == 0x67 )
      {
          header[0] = 0x00;
          header[1] = 0x00;
          header[2] = 0x00;
          header[3] = 0x01;
          header[4] = 0x67;

           memcopy(szCompleteH264SequenceHeader,header,5);
           memcopy(szCompleteH264SequenceHeader,
                   &psz_buf[RTP_HEADER_LEN+1],
                   POCKET_length - RTP_HEADER_LEN - 1 - B2_HEADER_LENGTH);

      }
}
```

FU-A(type 28) :

The FU header has the following format:

FU header			
1 bits	1 bits	1 bits	5 bits
S	E	R	TYPE

S: 1 bit
When set to one, the Start bit indicates the start of a fragmented NAL unit. When the following FU payload is not the start of a fragmented NAL unit payload, the Start bit is set to zero.

E: 1 bit
When set to one, the End bit indicates the end of a fragmented NAL unit, i.e., the last byte of the payload is also the last byte of the fragmented NAL unit. When the following FU payload is not the last fragment of a fragmented NAL unit, the End bit is set to zero.

R: 1 bit
The Reserved bit MUST be equal to 0 and MUST be ignored by the receiver.

Type: 5 bits
The NAL unit payload type.

The “FU-A pocket” should looked like this.

RTP Header (It’s detailed in previous chapter.)	NAL (1 byte)	FU header (1 byte)	FU Payload	
			(unprocessed segment of H.264 frame)	Header(0x000001B2) VIDEO_B2_FRAME (This B2 is attached when it’s the last segment of H.264 frame.)

When the type of NAL is 28, the received pocket is “FU-A Pocket”. We need further process to get complete “unprocessed segment of H.264 frame”.
We have to extend NAL and FU header first (bitwise operation), then append “unprocessed segment of H.264 frame “ later. The result will be “Complete segment of H.264 frame”.

Here is the concept code in C++.

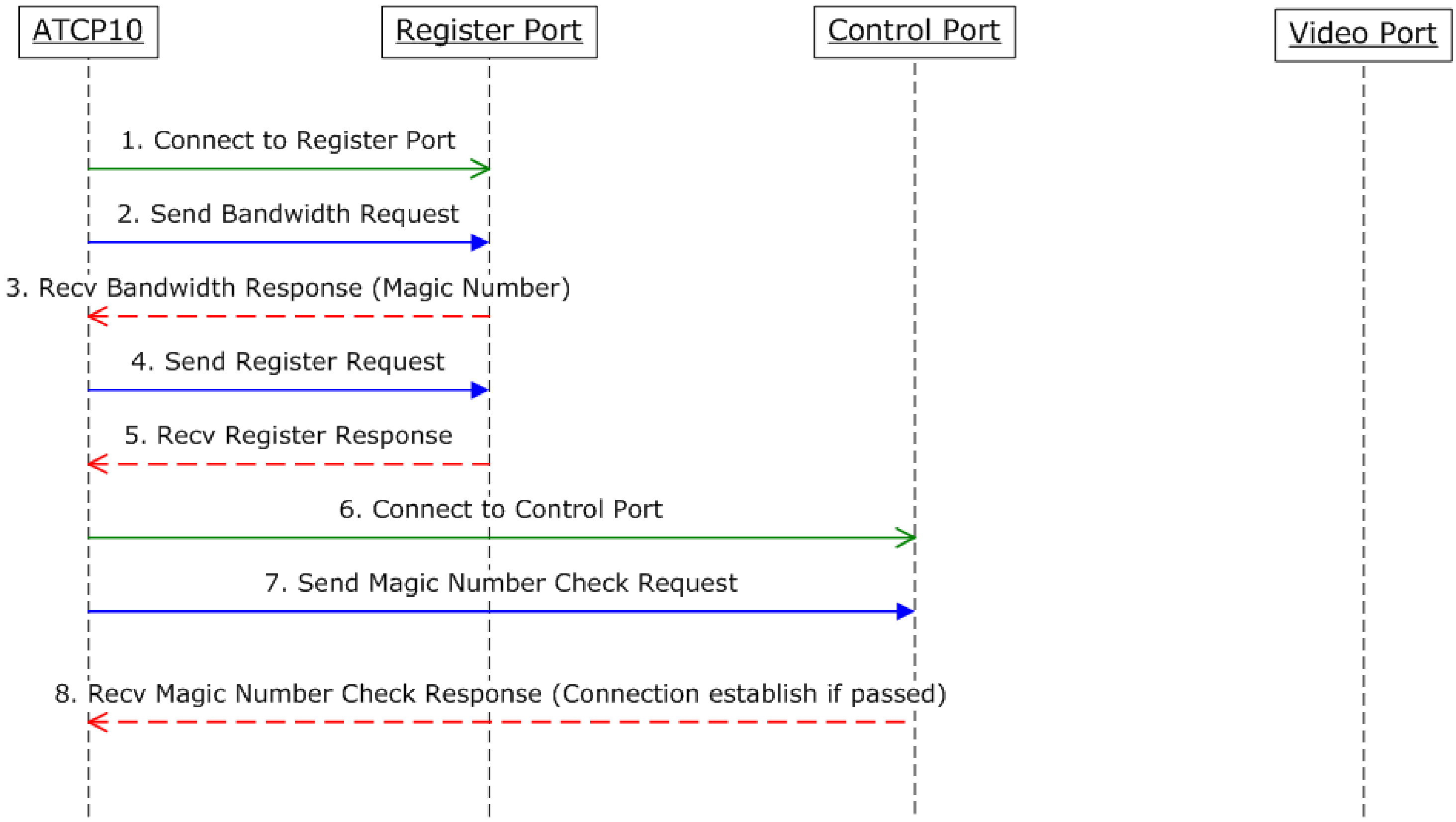
```
unsigned char szCompleteH264Segment[SEGMENT_SIZE];

if( h261_nal->TYPE == 28 )
{
    //copy fu header
    fuheader = (FU_HAEDER *)&psz_buf[RTP_HEADER_LEN+1];
    …
    // if the S of fu header is 1 , the segment is the first segment in this H.264 frame
    if( fuheader->S == 1 )
    {
        //first byte of this segment is 0x88, so it’ s a I frame
        if( (unsigned char)psz_buf[RTP_HEADER_LEN+2] == 0x88 )
        {
            // sometimes we put Sequence Header in front of I Frame for some decoder
            // I_Frame
            // Sequence + I Frame
            memcpy( frame.Buf+frame.Len, media.sequence_header, media.sequence_header_len);
            frame.Len += media.sequence_header_len;
        }
        else
        {

        }
        //
        header[0] = 0x00;
        header[1] = 0x00;
        header[2] = 0x00;
        header[3] = 0x01;
        header[4] = (h261_nal->NRI & 0x03) << 5 | fuheader->TYPE & 0x2F;
        memcopy(szCompleteH264Segment,header,5);
        // nal and fu header = 2Bytes
        memcopy(szCompleteH264Segment,
               &psz_buf[RTP_HEADER_LEN+2],
               POCKET_length - RTP_HEADER_LEN - 2);

        …
}
memcpy( frame.Buf+frame.Len, szCompleteH264Segment, POCKET_length - RTP_HEADER_LEN - 2 );
frame.Len += (POCKET_length - RTP_HEADER_LEN - 2);

//When Marker in RTP header is 1 , this is the last segment in H.264 frame
if( rtp_header->Marker == 1 )
{
    // Check If there is a B2 Header 00 00 01 B2
    if( GetRTPVideoFrameType( &frame.Buf[frame.Len-B2_FRAME_LEN]) == RTP_FRAME_B2 )
    {
    }
    else
    {
    }

}
```

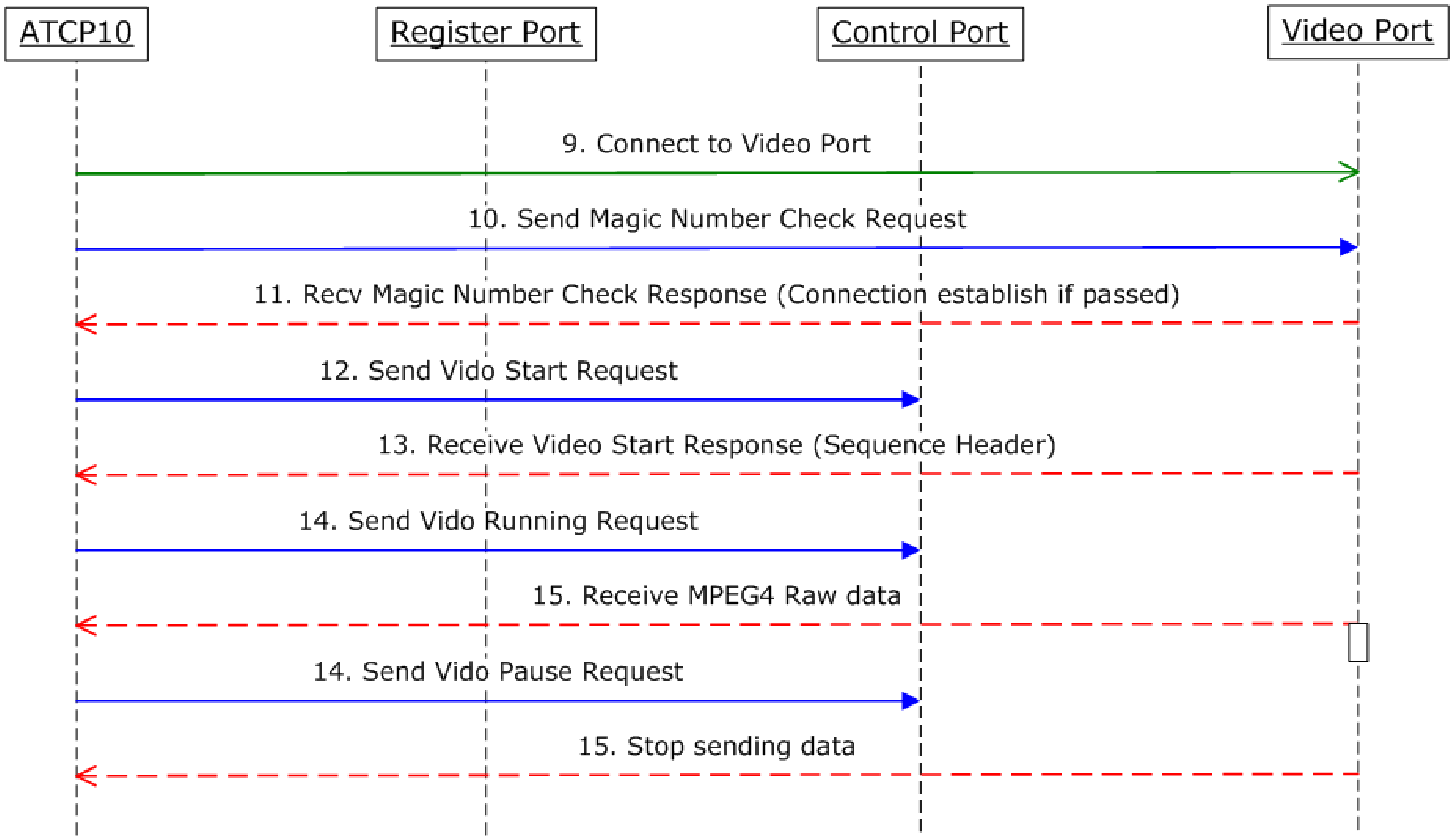
11

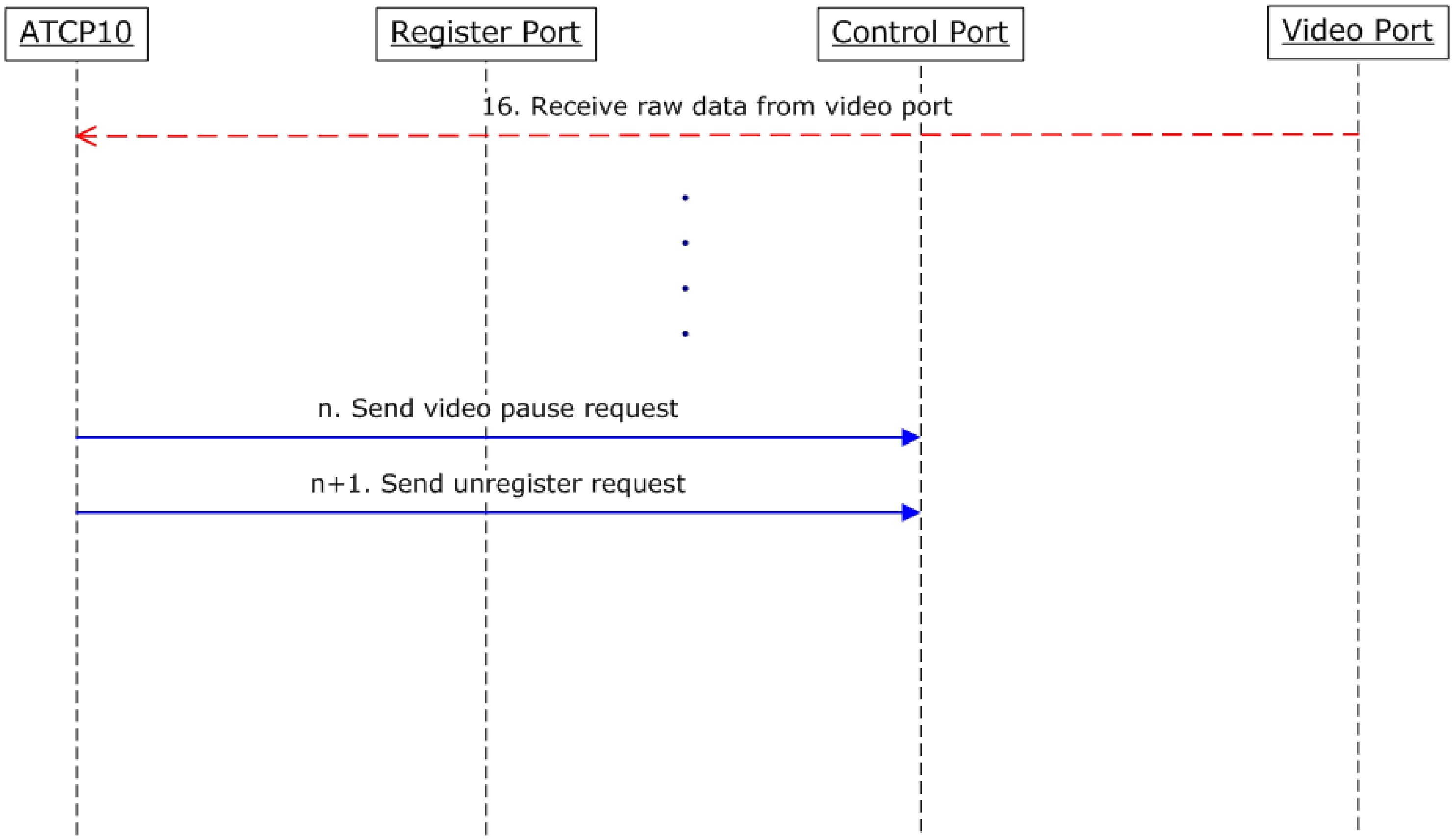
TCP and RTP/RTSP Packet Format

TCP v1.0 Packet

TCP v1.0 Video Connect Flow


ATCP10
Register Port
Control Port
Video Port
1. Connect to Register Port
2. Send Bandwidth Request
3. Recv Bandwidth Response (Magic Number)
4. Send Register Request
5. Recv Register Response
6. Connect to Control Port
7. Send Magic Number Check Request
8. Recv Magic Number Check Response (Connection establish if passed)


ATCP10
Register Port
Control Port
Video Port
9. Connect to Video Port
10. Send Magic Number Check Request
11. Recv Magic Number Check Response (Connection establish if passed)
12. Send Vido Start Request
13. Receive Video Start Response (Sequence Header)
14. Send Vido Running Request
15. Receive MPEG4 Raw data
14. Send Vido Pause Request
15. Stop sending data


ATCP10
Register Port
Control Port
Video Port
16. Receive raw data from video port
n. Send video pause request
n+1. Send unregister request

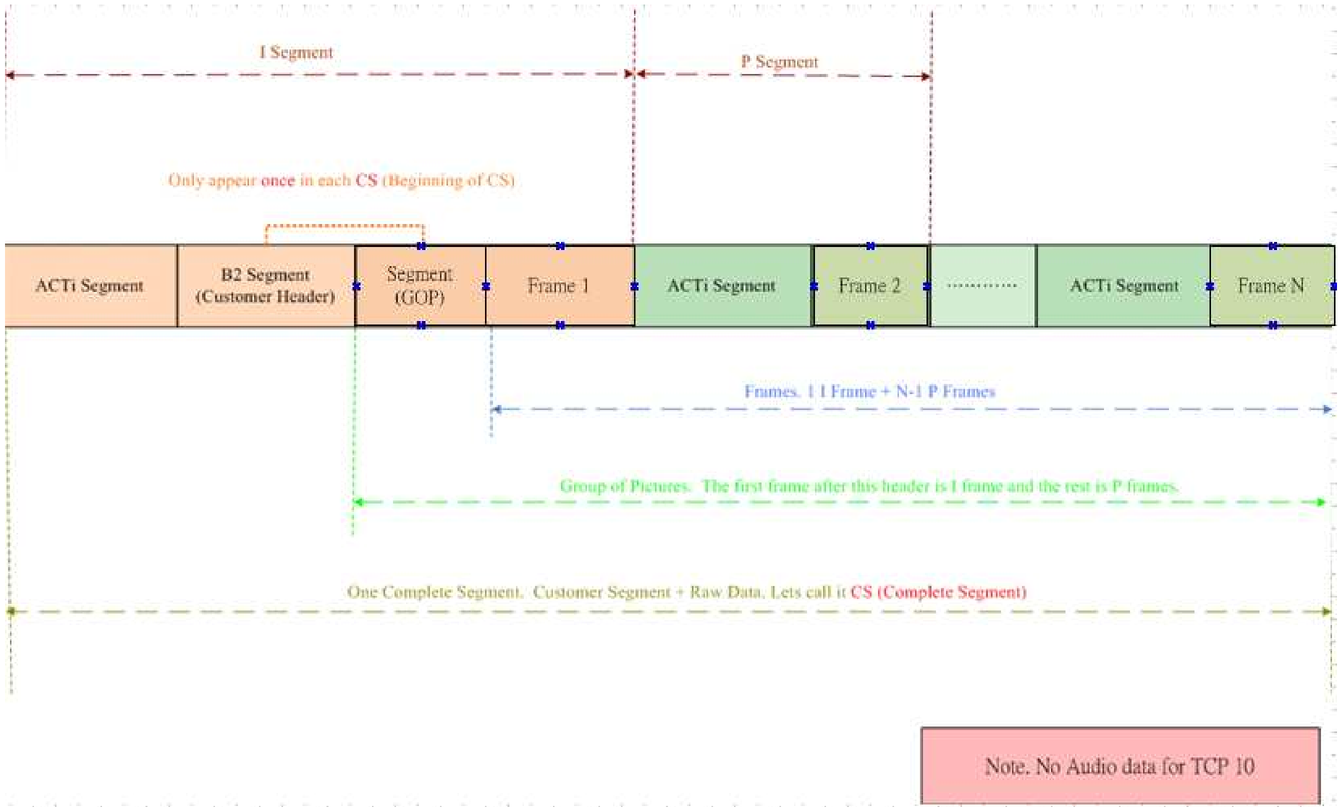
Note that:

1. Live check packet send between ATCP & Control port every constant time.

Disconnect steps

1. Disconnect register port
2. Do n and n+1 steps
3. Disconnect control port
4. Disconnect video port

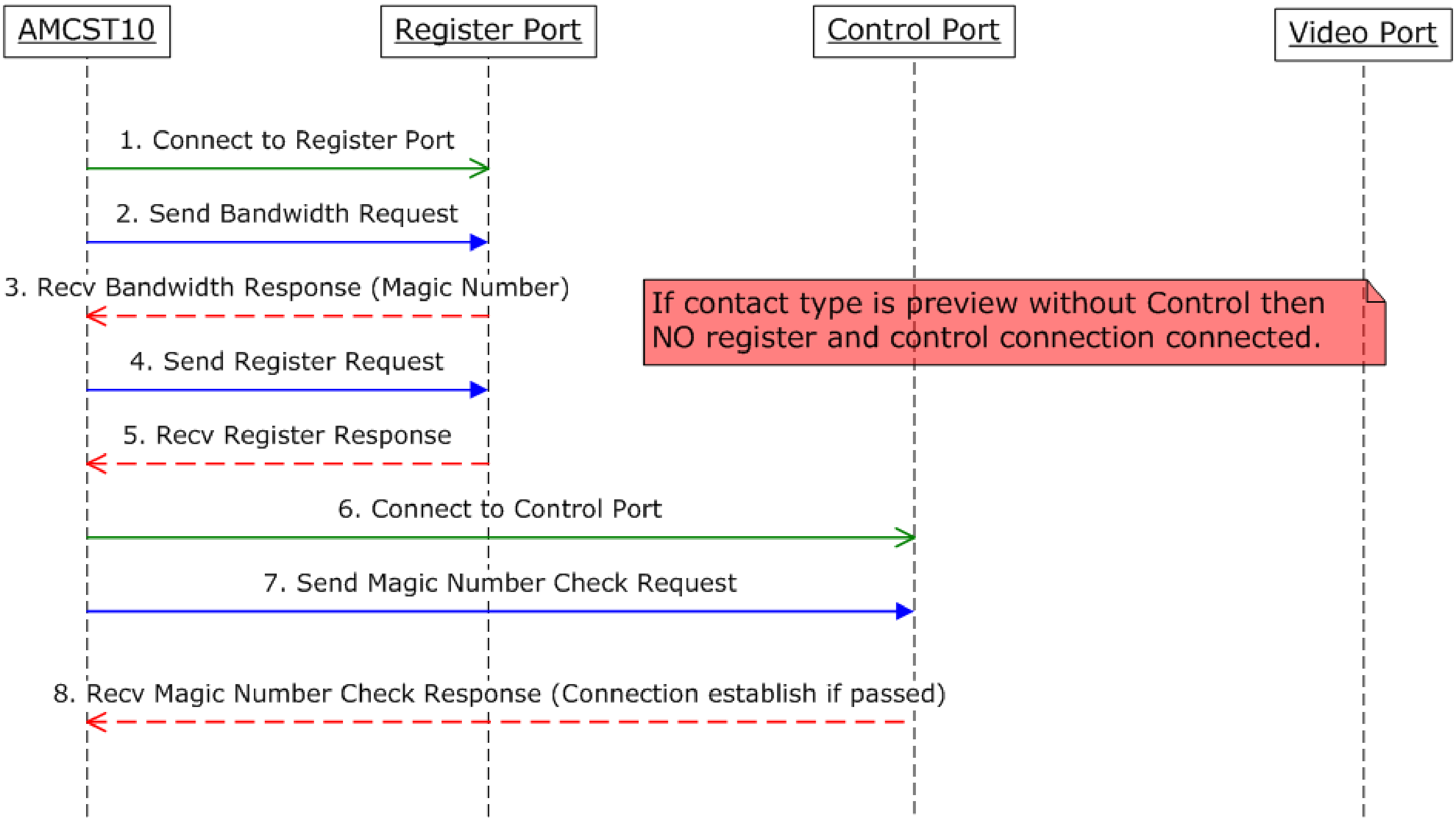
TCP v1.0 Video Packet Format

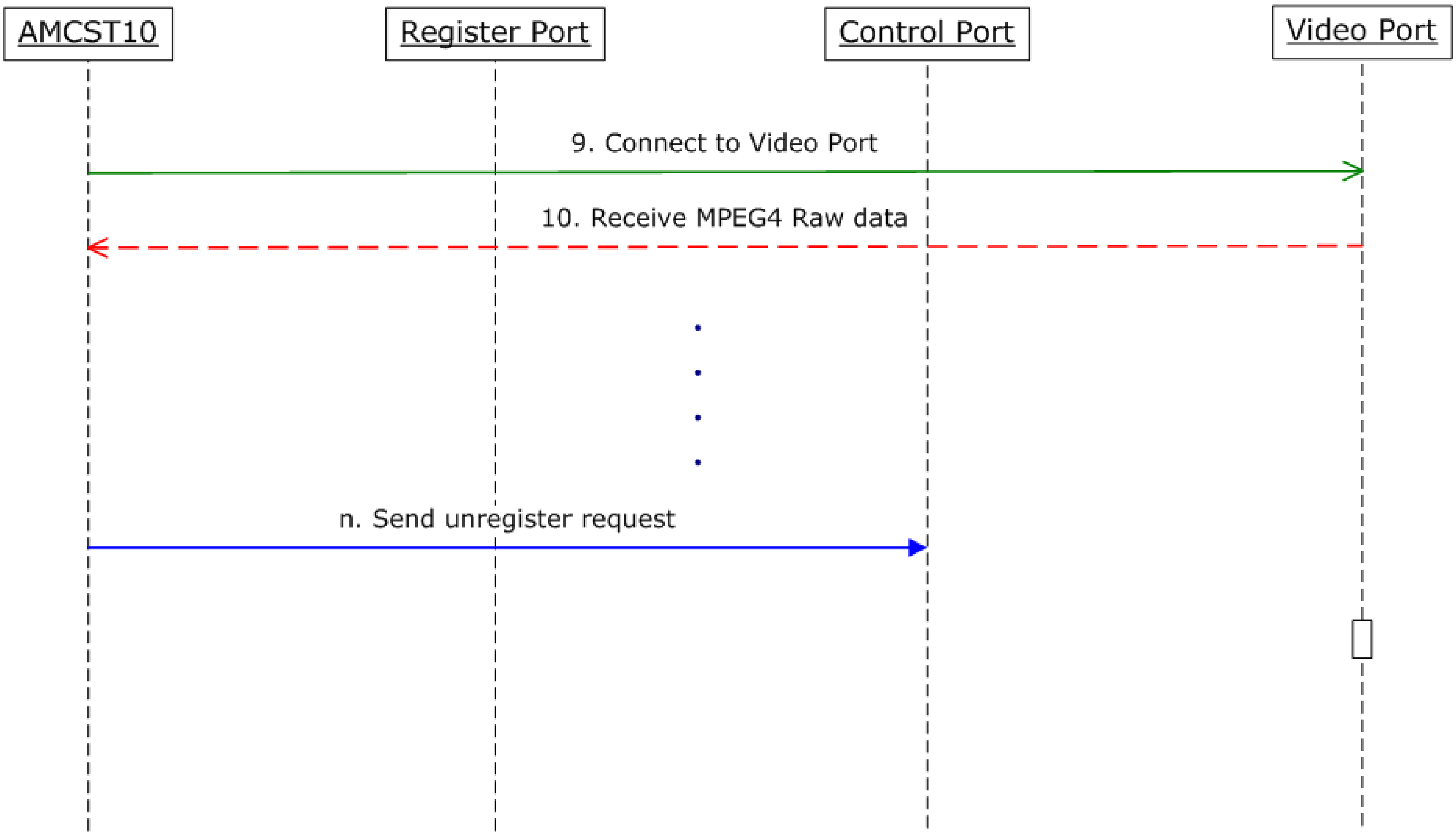

I Segment
P Segment
Only appear once in each CS (Beginning of CS)
ACTi Segment
B2 Segment
(Customer Header)
Segment
(GOP)
Frame 1
ACTi Segment
Frame 2
…………
ACTi Segment
Frame N
Frames. 1 I Frame + N-1 P Frames
Group of Pictures. The first frame after this header is I frame and the rest is P frames.
One Complete Segment. Customer Segment + Raw Data. Lets call it CS (Complete Segment)
Note. No Audio data for TCP 10

```
Segment
{
char  b2h[4];   // String "\x41\x43\x54\x67"
DWORD dwVersion;  // 0x00010022
DWORD dwLength;  // Data length
}
```

Multicast v1.0 Packet

Multicast v1.0 Video Connect Flow


AMCST10
Register Port
Control Port
Video Port
1. Connect to Register Port
2. Send Bandwidth Request
3. Recv Bandwidth Response (Magic Number)
4. Send Register Request
5. Recv Register Response
6. Connect to Control Port
7. Send Magic Number Check Request
8. Recv Magic Number Check Response (Connection establish if passed)
If contact type is preview without Control then NO register and control connection connected.



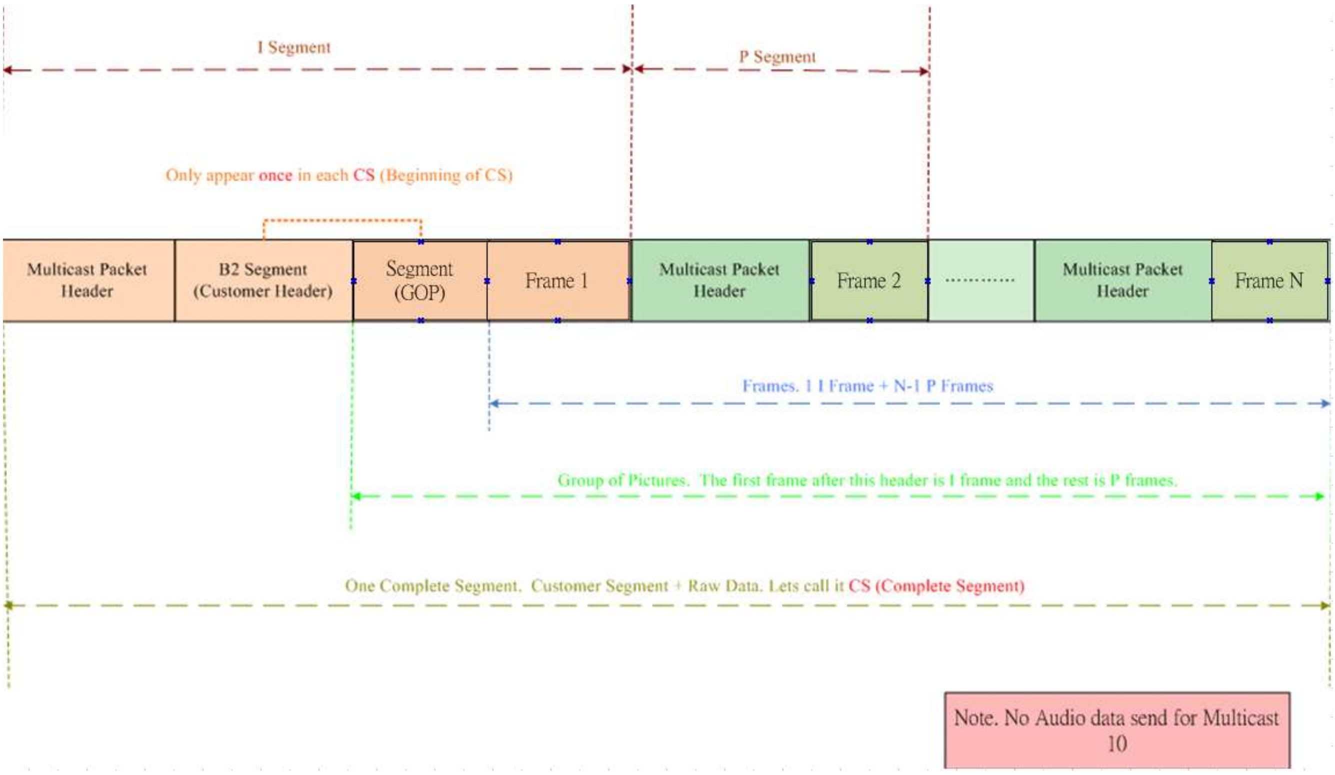
Note that:

Live check packet send between ATCP & Control port every constant time.

Disconnect steps

1. Disconnect register port
2. Do n.
3. Disconnect control port
4. Disconnect video port

Multicast v1.0 Video Packet Format


I Segment
P Segment
Only appear once in each CS (Beginning of CS)
Multicast Packet Header
B2 Segment (Customer Header)
Segment (GOP)
Frame 1
Multicast Packet Header
Frame 2
....
Multicast Packet Header
Frame N
Frames. 1 I Frame + N-1 P Frames
Group of Pictures. The first frame after this header is I frame and the rest is P frames.
One Complete Segment. Customer Segment + Raw Data. Lets call it CS (Complete Segment)
Note. No Audio data send for Multicast 10

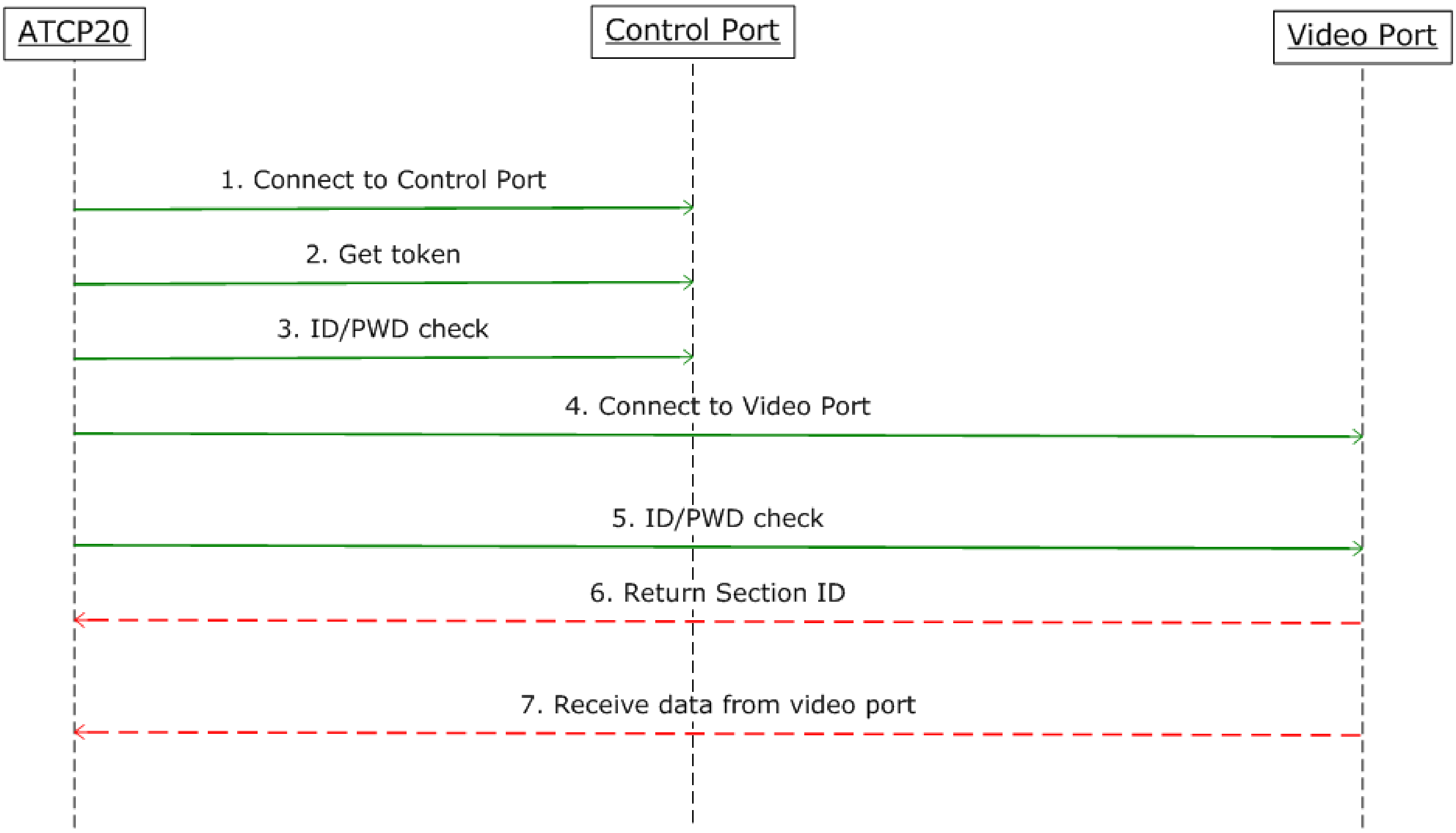
```
typedef struct tagMCPacketHead
{
unsigned char StreamId;
unsigned char StreamSubId;
unsigned char KeyPacket;
unsigned char TotalPacket;
unsigned char PacketNum;
unsigned char FrameCheckSum;
unsigned char Resolution;
unsigned char Fps;
unsigned int  FrameNum;
unsigned int  FrameLen;
} MCPacketHead;
```

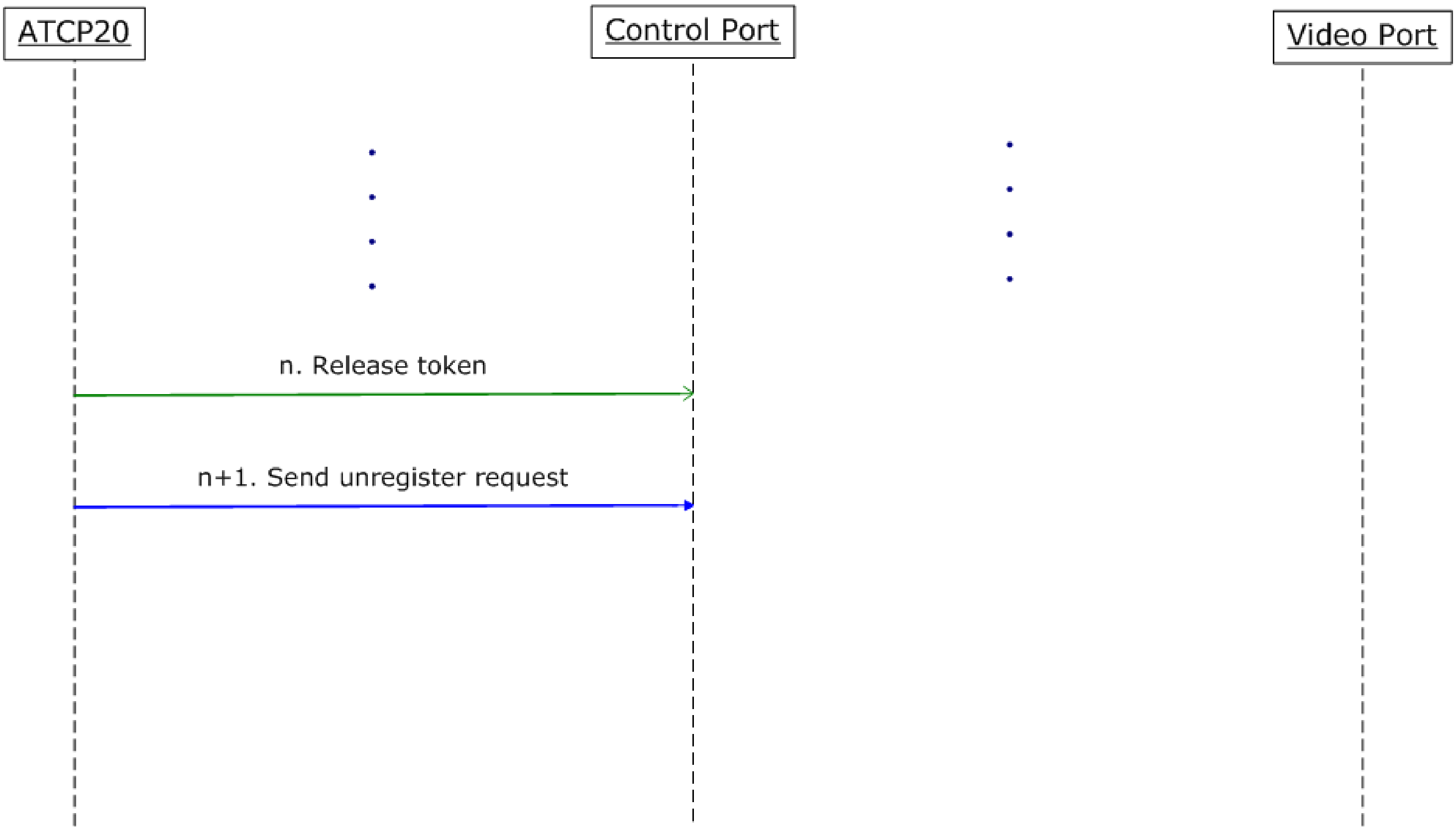
Important Note:

1. Key packet attribute is very important to determine the last packet of the frame.
2. Only key packet has both FPS and Resolution information.

TCP v2.0 Packet

TCP 2.0 Video Connect Flow

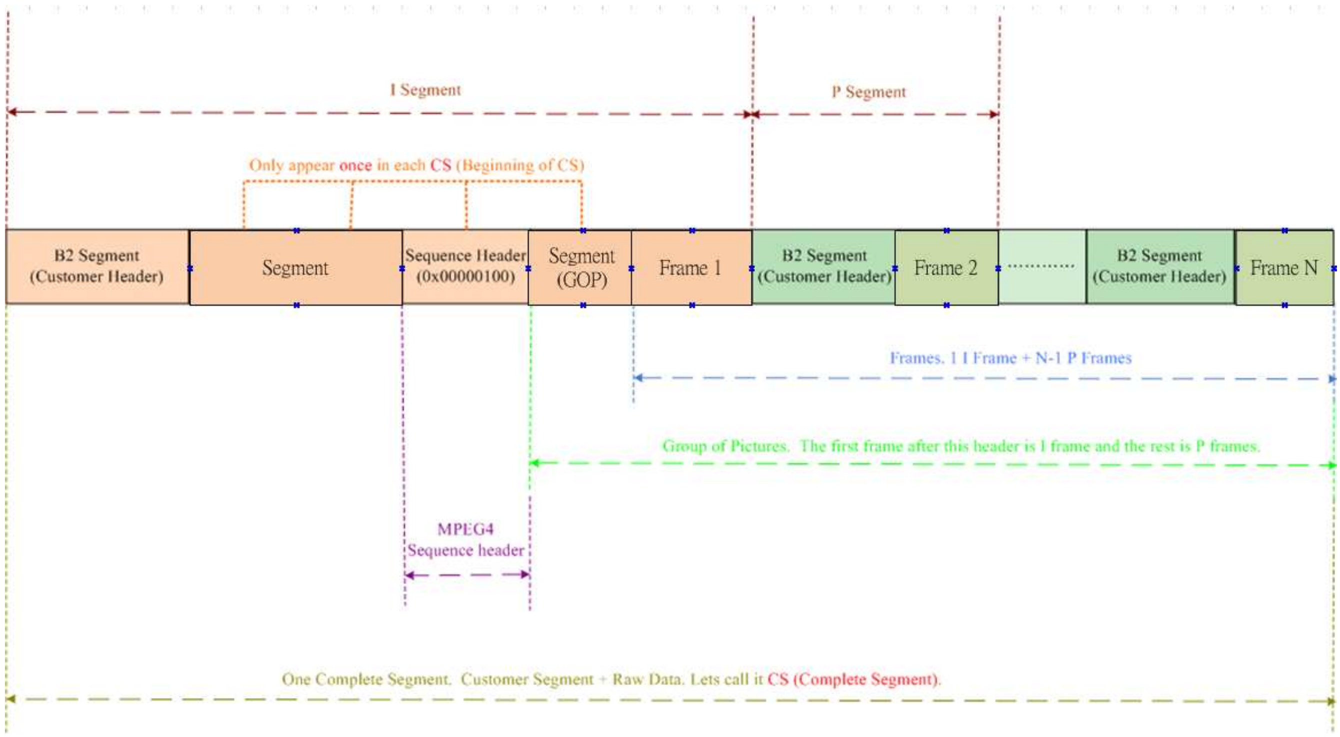




Disconnect steps

1. Do n and n+1 steps
2. Disconnect control port
3. Disconnect video port

TCP 2.0 Video Packet Format


I Segment
P Segment
Only appear once in each CS (Beginning of CS)
B2 Segment (Customer Header)
Segment
Sequence Header (0x00000100)
Segment (GOP)
Frame 1
B2 Segment (Customer Header)
Frame 2
…………
B2 Segment (Customer Header)
Frame N
Frames. 1 I Frame + N-1 P Frames
Group of Pictures. The first frame after this header is I frame and the rest is P frames.
MPEG4 Sequence header
One Complete Segment. Customer Segment + Raw Data. Lets call it CS (Complete Segment).

Composition of data:

B2 Frame (see the detail in chap.8)

(B2 header is 44 Byte for MPEG4/MJPEG/H264 Frames , AUDIO_B2 header is 28 Byte)

VIDEO_B2_FRAME	
B2_HEADER	
0x000001B2	User Data

```
typedef struct {
 B2_HEADER    header;
 PRIVATE_DATA   prdata;
} VIDEO_B2_FRAME; //44 bytes (details in chapter 8)
```

AUDIO_B2	
B2_HEADER	
0x000001B2	User Data

```
typedef struct {
 B2_HEADER Head;
 struct timeval TS; //timestamp
 unsigned char Rsvd[8];
} AUDIO_B2_FRAME;//28 bytes (details in chapter 8)
```

Bitstream Data			
Header	Data	Sequence Header	Sequence Data
		0x00000100	17 bytes

I-Frame Data		
Header	Data	Frame Data

P-Frame Data	
Header	Frame Data

Audio Data
(PCM data)
N Byte

TCP :

(a) MPEG4 I Frame composite

<table>
<tr><td colspan="2">VIDEO_B2_FRAME</td><td colspan="3">Bitstream Data</td><td colspan="3">I-Frame Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Sequence</td><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Frame data</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>0x00000100</td><td>N Byte</td></tr>
</table>

(b) MPEG4 P Frame composite

<table>
<tr><td colspan="2">VIDEO_B2_FRAME</td><td colspan="2">P-Frame Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td rowspan="2">Header</td><td>Frame data</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>N Byte</td></tr>
</table>

(c) Audio Frame composite

<table>
<tr><td colspan="2">AUDIO_B2</td><td>Audio Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td>(PCM data)</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>N Byte</td></tr>
</table>

Multicast :

(a) MPEG4 I Frame

<table>
<tr><td rowspan="3">Multicast Header</td><td colspan="2">VIDEO_B2_FRAME</td><td colspan="3">Bitstream Data</td><td colspan="3">I-Frame Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Sequence</td><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Frame data</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>0x00000100</td><td>N Byte</td></tr>
</table>

(b) MPEG4 P Frame

<table>
<tr><td rowspan="3">Multicast Header</td><td colspan="2">VIDEO_B2_FRAME</td><td colspan="2">P-Frame Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td rowspan="2">Header</td><td>Frame data</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>N Byte</td></tr>
</table>

(c) Audio Frame

<table>
<tr><td rowspan="3">Multicast Header</td><td colspan="2">AUDIO_B2</td><td>Audio Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td>(PCM data)</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>N Byte</td></tr>
</table>

Exported Struct

Media Connection Configuration :

```
typedef struct structural_MEDIA_CONNECTION_CONFIG2
{
 int  ContactType;
 unsigned char ChannelNumber; // For URL Command CHANNEL tag, when set it to 0,
                               URL command won't bring CHANNEL tag, or the URL
                               command will bring CHANNEL=ChannelNumber tag  )
 unsigned char TCPVideoStreamID; // 0 based to specify video track, value 0 to
                                 255 for 1 to 256 video track
 unsigned char RTPVideoTrackNumber; // set it to 0, ARTP will use 1st video track,
                                    1 to 255 is for specify video track
 unsigned char RTPAudioTrackNumber; // set it to 0, ARTP will use 1st audio track,
                                    1 to 255 is for specify audio track
 char          UniCastIP[256];
 char          MultiCastIP[16];
 char          PlayFileName[256];
 char          UserID[64];
 char          Password[64];
 unsigned long RegisterPort;
 unsigned long StreamingPort;
 unsigned long ControlPort;
 unsigned long MultiCastPort;
 unsigned long SearchPortC2S;
 unsigned long SearchPortS2C;
 unsigned long HTTPPort;
 unsigned long RTSPPort;
 unsigned long Reserved1;
 unsigned long Reserved2;

 unsigned short    ConnectTimeOut;
 unsigned short    EncryptionType;
}MEDIA_CONNECTION_CONFIG2;
```

Media Video Configuration :

```
typedef struct structural_MEDIA_VIDEO_CONFIG2
{
 short dwEncoder;           // 1:MPEG4 4:MPEG4 5:H264
 short dwTvStander;         // 0:NTSC 1:PAL
 short dwVideoResolution;   // See the definition above
 short dwBitsRate;          // See the definition above
 short dwQuality;           // 0 ~ 100 : Low ~ High
 short dwVideoBrightness;   // 0 ~ 100 : Low ~ High
 short dwVideoContrast;     // 0 ~ 100 : Low ~ High
 short dwVideoSaturation;   // 0 ~ 100 : Low ~ High
 short dwVideoHue;          // 0 ~ 100 : Low ~ High
 short dwFps;               // 0 ~ 30 frame pre second
} MEDIA_VIDEO_CONFIG2;
```

Media Port Information :

```
typedef struct structural_MEDIA_PORT_INFO    // Device port info.
{
 unsigned long PORT_HTTP;                // HTTP Port
 unsigned long PORT_SearchPortC2S;       // Search Port 1
 unsigned long PORT_SearchPortS2C;       // Search Port 2
 unsigned long PORT_Register;            // Register Port
 unsigned long PORT_Control;             // Control Port
 unsigned long PORT_Streaming;           // Streaming Port
 unsigned long PORT_Multicast;           // Multicast Port
 unsigned long PORT_RTSP;                // RTSP Port
} MEDIA_PORT_INFO;
```

Media Render Information

```
typedef struct structural_MEDIA_RENDER_INFO
{
 int   RenderInterface;              // Reserve, in the future this
                                     // parameter meaning DGDI or DDRAW
 HWND  hWnd;                             // The handle of drawing window
 RECT  rect;                             // rect. info. of drawing window.
} MEDIA_RENDER_INFO;
```

Media Motion Information

```
#define MD_REGION_SIZE 4
typedef struct structural_MEDIA_MOTION_INFO_EX
{
 DWORD dwEnable;
 DWORD dwRangeCount;
 DWORD dwRange[MD_REGION_SIZE][4];
 DWORD dwSensitive[MD_REGION_SIZE];
 DWORD dwTime[MD_REGION_SIZE];
 DWORD dwThreshold[MD_REGION_SIZE];
 DWORD bEnable[MD_REGION_SIZE];
} MEDIA_MOTION_INFO_EX;
```

Raw File Record Information

```
typedef struct structural_MP4FILE_RECORD_INFO
{
 unsigned      long tBeginTime;
 unsigned      long tEndTime;
 BYTE          btTimeZone;
 DWORD         dwGOP;
 DWORD         dwFrameCount;
 ULONGLONG     FileSize;
} MP4FILE_RECORD_INFO;
```

Time Zone

0 : GMT-12
1 : GMT-11
2 : GMT-10
3 : GMT-09
4 : GMT-08
5 : GMT-07
6 : GMT-06
7 : GMT-05
8 : GMT-04
9 : GMT-03
10 : GMT-02
11 : GMT-01
12 : GMT+00
13 : GMT+01
14 : GMT+02
15 : GMT+03
16 : GMT+04
17 : GMT+05
18 : GMT+06
19 : GMT+07
20 : GMT+08
21 : GMT+09
22 : GMT+10
23 : GMT+11
24 : GMT+12
25 : GMT+13
32 : GMT-9:30
33:GMT-4:30
34:GMT-3:30
35:GMT+3:30
36:GMT+4:30
37 :GMT+5:30
38:GMT+5:45
39:GMT+6:30
40:GMT+9:30
41:GMT+11:30
42:GMT+12:45

DI Notify

```
typedef struct structural_NOTIFY_DI
{
 HANDLE    DIEvent;                        // [IN] Event handle
 BYTE DI;                                  // [OUT] Digital input
}NOTIFY_DI;
```

Time Code Notify

```
typedef struct structural_NOTIFY_TIMECODE
{
 HANDLE    TimeCodeEvent;                  // [IN] Event handle
 DWORD     dwTimeCode;                     // [OUT] Time code
}NOTIFY_TIMECODE;
```

Raw Data Refresh Notify

```
typedef struct structural_NOTIFY_RAWDATAREFRESH
{
 HANDLE   RawDataRefreshEvent;            // [IN] Event handle
 void*    pBuffer;                        // [OUT] Buffer pointer
 int      nFillLength;                    // [IN/OUT] Buffer length
}NOTIFY_RAWDATA_REFRESH;
```

Video Status Notify

```
typedef struct structural_NOTIFY_VIDEOSTATUS
{
 HANDLE   VideoLossEvent;                 // [IN] Event handle
 HANDLE   VideoRecoveryEvent;             // [IN] Event handle
}NOTIFY_VIDEO_STATUS;
```

Network Loss Notify

```
typedef struct structural_NOTIFY_NETWORKLOSS
{
 HANDLE   NetworkLossEvent;               // [IN] Event handle
}NOTIFY_NETWORK_LOSS;
```

Motion Detection Notify

```
typedef struct structural_NOTIFY_MOTIONDETECTION
{
 HANDLE   MotionDetectionEvent;           // [IN] Event handle
 BYTE     MotionDetection;                // [OUT] Motion detection info.
}NOTIFY_MOTION_DETECTION;
```

Image Refresh Notify

```
typedef struct structural_NOTIFY_IMAGE_REFRESH
{
 HANDLE   ImageRefreshEvent;              // [IN] Event handle
 void*    pImage;                         // [OUT] Buffer pointer
 int      nFillLength;                    // [IN/OUT] Buffer length
}NOTIFY_IMAGE_REFRESH;
```

After Render Notify

```
typedef struct structural_NOTIFY_AFTER_RENDER
{
 HANDLE   AfterRenderEvent;               // [IN] Event handle
}NOTIFY_AFTER_RENDER;
```

Resolution Change Notify

```
typedef struct structural_NOTIFY_RESOLUTION_CHANGE
{
 HANDLE   ResolutionChangeEvent;         // [IN] Event handle
 int      nResolution;                   // [OUT] Resolution
}NOTIFY_RESOLUTION_CHANGE;
```

Resolution Map

In this chapter, new megapixel resolution has been added.

```
#define XNTSC720x480       0         //0x00  0000 0000
#define XNTSC352x240       1         //0x01  0000 0001
#define XNTSC160x112       2         //0x02  0000 0010
#define XPAL720x576        3         //0x03  0000 0011
#define XPAL352x288        4         //0x04  0000 0100
#define XPAL176x144        5         //0x05  0000 0101
#define XNTSC176x120       6         //0x06  0000 0110
#define XNTSC640x480       64        //0x40  0100 0000
#define XPAL640x480        192       //0xC0  1100 0000
#define XNTSC1280x720       65       //0x41  0100 0001
#define XNTSC1280x960       66       //0x42  0100 0010
#define XNTSC1280x1024      67       //0x43  0100 0011
#define XNTSC1600x1200      68       //0x44  0100 0100
#define XNTSC1920x1080      69       //0x45  0100 0101
#define XNTSC2032x1920     72       //0x48  0100 1000
#define XNTSC320x240       70       //0x46  0100 0110
#define XNTSC160x120       71       //0x47  0100 0111
#define XNTSC2592x1944     75       //0x4B  0100 1011
#define XNTSC2048x1536     76       //0x4C  0100 1100
```

RS232 Data Refresh Notify

```
typedef struct structural_NOTIFY_RS232DATA_REFRESH
{
 HANDLE   RS232DataRefreshEvent;         // Event handle
 void*    pBuffer;                       // [OUT] Buffer pointer
 int      nFillLength;                   // [IN/OUT] Buffer length
}NOTIFY_RS232DATA_REFRESH;
```

Digital Input Default Value

```
#define DI_DEFAULT_IS_LOW       0x00         // Digital Input Default is Low
#define DI_DEFAULT_IS_HIGH      0x03         // Digital Input Default is High
```

Digital Output Value

```
#define DO_OUTPUT_1             0x01         // Digital Output 1st
#define DO_OUTPUT_2             0x02         // Digital Output 2nd
#define DO_OUTPUT_BOTH          0x03         // Digital Output Both 1st & 2nd
#define DO_OUTPUT_CLEAN        0x00          // Clen up Digital Output
```

RS232 Setting

```
#define NET_N81    0x00     //N, 8, 1
#define NET_O81    0x08     //Odd, 8, 1
#define NET_E81    0x18     //Even, 8, 1
#define NET_8N1    0x81     //N, 8, 1
#define NET_8O1    0x89     //Odd, 8, 1
#define NET_8E1    0x85     //Even, 8, 1
#define NET_8N2    0x82     //8 Data bits, No parity check, 2 Stop bit
#define NET_8O2    0x8A    //8 Data bits, Odd parity check, 2 Stop bit
#define NET_8E2    0x86    //8 Data bits, Even parity check, 2 Stop bit
#define NET_7N2    0x72     //7 Data bits, No parity check, 2 Stop bit
#define NET_7O2    0x7A     //7 Data bits, Odd parity check, 2 Stop bit
#define NET_7E2    0x76     //7 Data bits, Even parity check, 2 Stop bit
```

Play Rate

```
enum PLAY_RATES                              // Play rate
{
 RATE_0_5,                              // 1/2 Speed
 RATE_1_0,                              // 1.0 Speed
 RATE_2_0,                              // 2.0 Speed
 RATE_4_0,                              // 4.0 Speed
 RATE_8_0                               // 8.0 Speed
};
```

Contact Type

```
enum {                                          //Contact Type
 CONTACT_TYPE_UNUSE = 0,                        //not used
 CONTACT_TYPE_UNICAST_WOC_PREVIEW=1,           //without control port
 CONTACT_TYPE_MULTICAST_WOC_PREVIEW,           //without control port
 CONTACT_TYPE_RTSP_PREVIEW,
 CONTACT_TYPE_CONTROL_ONLY,
 CONTACT_TYPE_UNICAST_PREVIEW,
 CONTACT_TYPE_MULTICAST_PREVIEW,
 CONTACT_TYPE_PLAYBACK,
 CONTACT_TYPE_CARD_PREVIEW,
 CONTACT_TYPE_MULTIPLE_PLAYBACK,
 CONTACT_TYPE_HTTP_RTSP_WOC_PREVIEW,            //without control port
 CONTACT_TYPE_HTTP_RTSP_PREVIEW,
 CONTACT_TYPE_HTTP,
 CONTACT_TYPE_RTSP_RTPUDP_PREVIEW,
 CONTACT_TYPE_RTSP_RTPUDP_WOC_PREVIEW,
 CONTACT_TYPE_HTTP_WOC_PREVIEW,                 //without control port
 CONTACT_TYPE_HTTP_PREVIEW,
 CONTACT_TYPE_HTTP_CONTROL_ONLY,
 CONTACT_TYPE_HTTP_MESSAGE,             //dual session;listen and POST Message
 CONTACT_TYPE_HTTP_REMOTE_PLAYBACK,     // Remote Playback (search/play)
 CONTACT_TYPE_HTTP_AUDIO_TRANSFER,      // Send Audio to Device
 CONTACT_TYPE_PLAYBACK_AVI = 60,
 CONTACT_TYPE_MAX,
}CONTACT_TYPE;
```

RS232 Baud Rate

```
enum RS232_BAUD_RATE                          // RS232 BaudRate
{
 BAUD_RATE_1200BPS,                      // 1200 BPS
 BAUD_RATE_2400BPS,                      // 2400 BPS
 BAUD_RATE_4800BPS,                      // 4800 BPS
 BAUD_RATE_9600BPS,                      // 9600 BPS
 BAUD_RATE_19200BPS,                     // 19200 BPS
 BAUD_RATE_38400BPS,                     // 38400 BPS
 BAUD_RATE_57600BPS,                     // 57600 BPS
 BAUD_RATE_115200BPS,                    // 115200 BPS
 BAUD_RATE_230400BPS                     // 230400 BPS
};
```

Bit Rate

```
enum BITRATE_TYPES          /** Bitrate Types */
{
 BITRATE_28K,               ///< #0# - 28K Bits per second
 BITRATE_56K,               ///< #1# - 56K Bits per second
 BITRATE_128K,              ///< #2# - 128K Bits per second
 BITRATE_256K,              ///< #3# - 256K Bits per second
 BITRATE_384K,              ///< #4# - 384K Bits per second
 BITRATE_500K,              ///< #5# - 500K Bits per second
 BITRATE_750K,              ///< #6# - 750K Bits per second
 BITRATE_1000K,             ///< #7# - 1M Bits per second
 BITRATE_1200K,             ///< #8# - 1.2M Bits per second
 BITRATE_1500K,             ///< #9# - 1.5M Bits per second
 BITRATE_2000K,             ///< #10# - 2M Bits per second
 BITRATE_2500K,             ///< #11# - 2.5M Bits per second
 BITRATE_3000K,             ///< #12# - 3M Bits per second
 BITRATE_3500K,             ///< #12# - 3.5M Bits per second
 BITRATE_4000K,             ///< #12# - 4M Bits per second
 BITRATE_4500K,             ///< #12# - 4.5M Bits per second
 BITRATE_5000K,             ///< #12# - 5M Bits per second
 BITRATE_5500K,             ///< #12# - 5.5M Bits per second
 BITRATE_6000K,             ///< #12# - 6M Bits per second
};
```

Codec Type

```
enum CODEC_TYPES            /** CODEC Types */
{
 XVIDCODEC,                 ///< #0# - XVID - using XVIDCODEC
 WISCODEC,                  ///< #1# - WIS - using WISCODEC
 P51CODEC,                  ///< #2# - PCI5100 - using P51CODEC
 IPPCODEC,                  ///< #3# - IPPCODEC - using IPPCODEC
 MJPEGCODEC,
 IH264CODEC
};
```

File Write Type

```
enum FILE_WRITER_TYPES                             // File writer types
{
 FRAW,                                        // Record by *.Raw File - using FRAW
 FAVI                                         // Record by *.Avi File - using FAVI
};
```

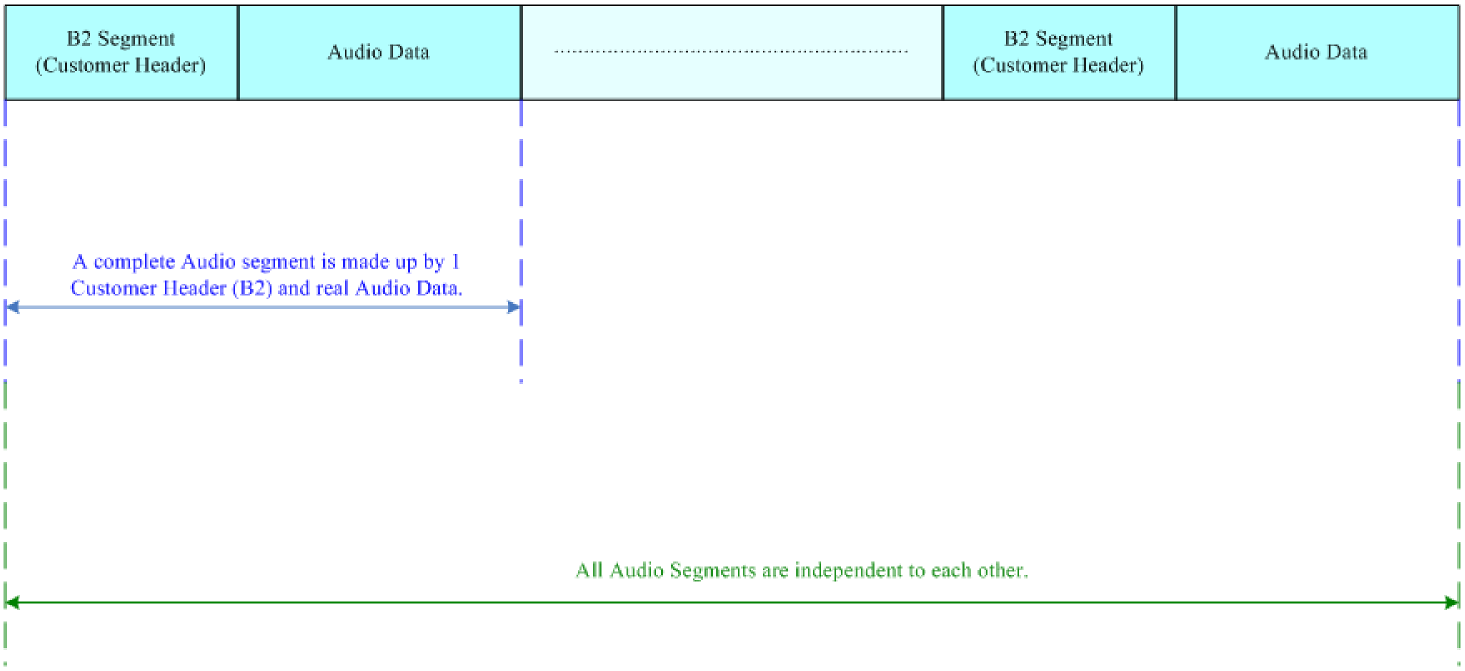
Render Type

```
enum RENDER_TYPES                                  // Render interface types
{
 DGDI,                                        // Windows GDI for render
 DDRAW                                        // Direct Draw for render
};
```

Device Type

```
enum DEVICE_TYPE                                   // Device Type
{
 Type_None,                                   // None type
 Type_StandAlong,                             // Stand along
 Type_RackMount,                              // Rack Mount
 Type_Blade                                   // Blade
};
```

TCP v2.0 Audio Packet Format

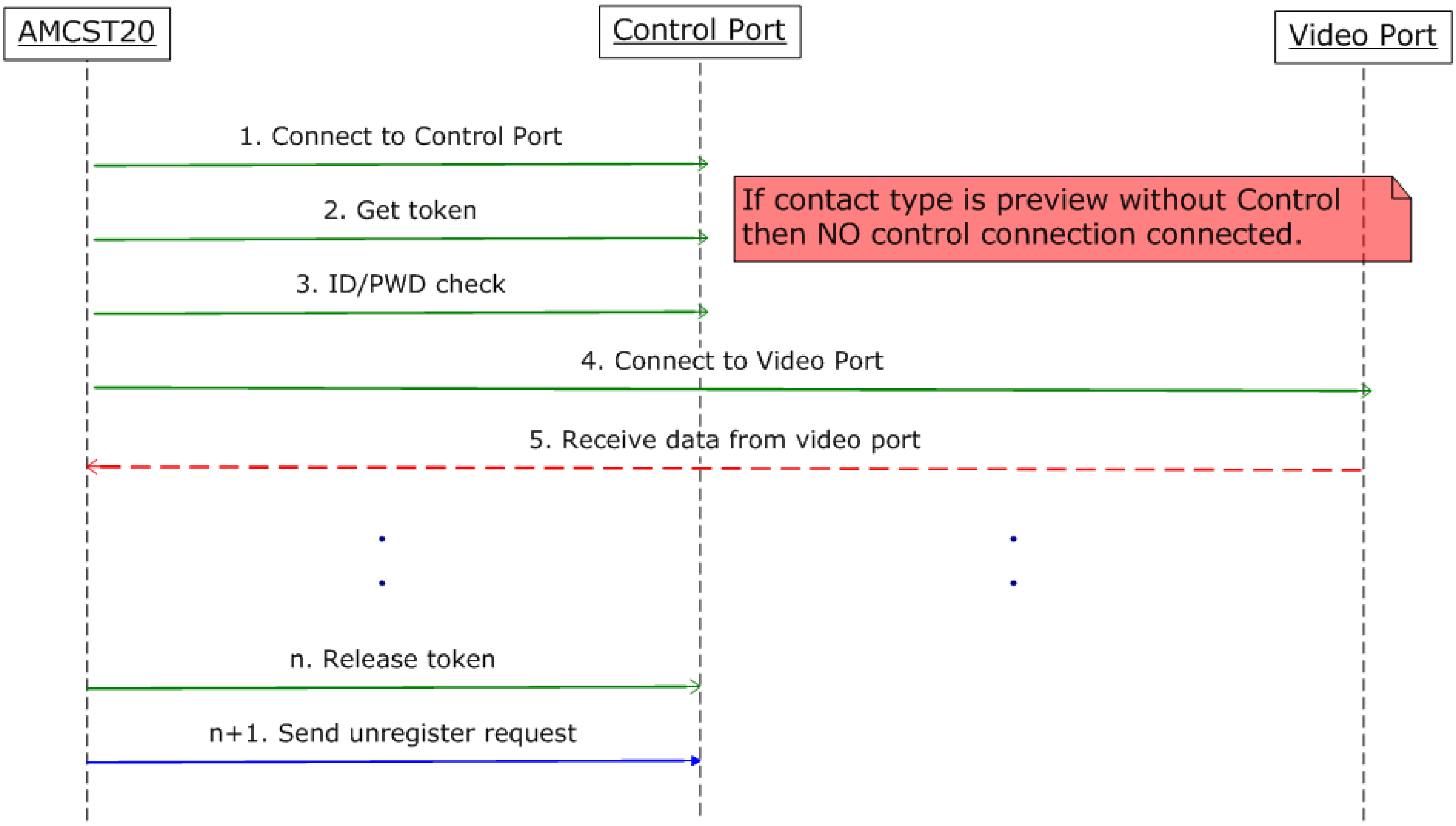

B2 Segment (Customer Header)
Audio Data
……………………………………………………
B2 Segment (Customer Header)
Audio Data
A complete Audio segment is made up by 1 Customer Header (B2) and real Audio Data.
All Audio Segments are independent to each other.

Note. Video server will send Audio & Video data in random order.

```
typedef struct {
 B2_HEADER Head;
 struct timeval TS; //timestamp
 unsigned char Rsvd[8];
} AUDIO_B2_FRAME;//28 bytes (details in chapter 8)
```

Multicast v2.0 Packet

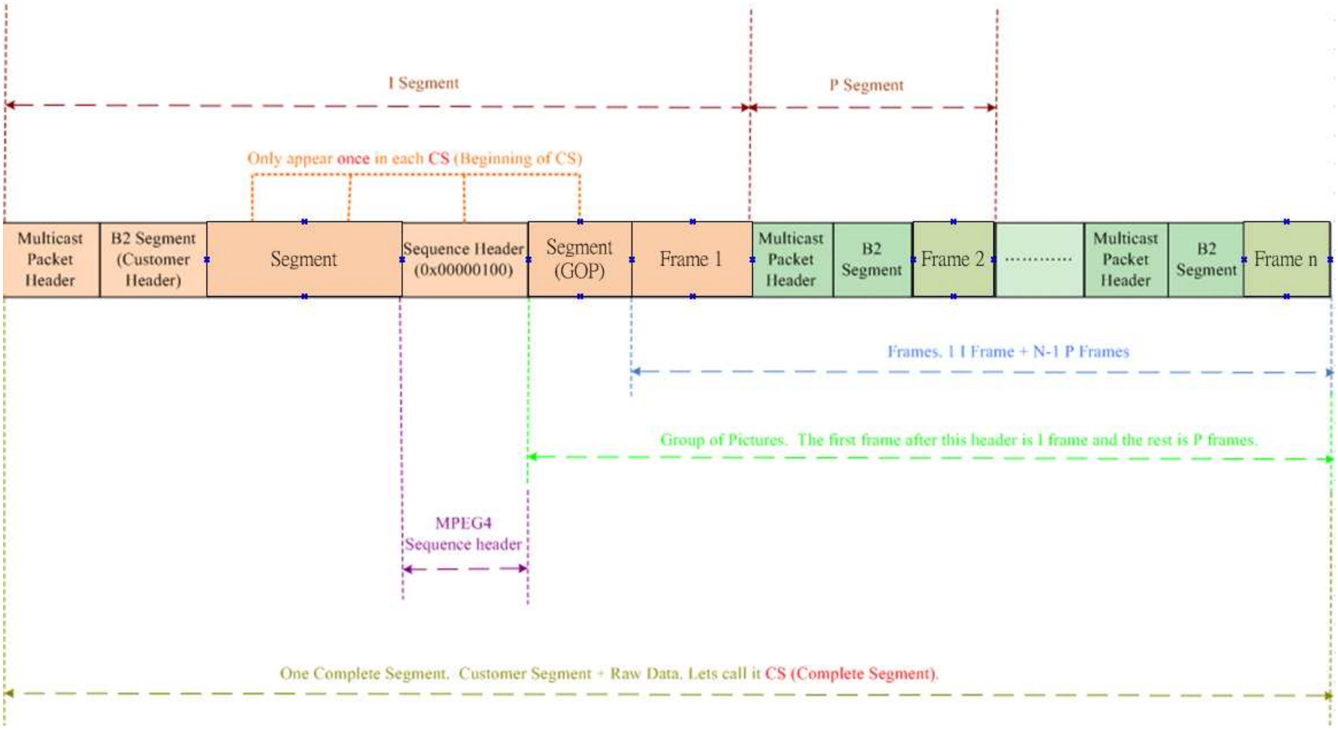
Multicast v2.0 Video Connect Flow


AMCST20
Control Port
Video Port
1. Connect to Control Port
2. Get token
3. ID/PWD check
If contact type is preview without Control then NO control connection connected.
4. Connect to Video Port
5. Receive data from video port
n. Release token
n+1. Send unregister request

Disconnect steps

1. Do n.
2. Disconnect control port
3. Disconnect video port

Multicast v2.0 Video Packet Format


I Segment
P Segment
Only appear once in each CS (Beginning of CS)
Multicast Packet Header
B2 Segment (Customer Header)
Segment
Sequence Header (0x00000100)
Segment (GOP)
Frame 1
Multicast Packet Header
B2 Segment
Frame 2
…………
Multicast Packet Header
B2 Segment
Frame n
Frames. 1 I Frame + N-1 P Frames
Group of Pictures. The first frame after this header is I frame and the rest is P frames.
MPEG4 Sequence header
One Complete Segment. Customer Segment + Raw Data. Lets call it CS (Complete Segment).

```
typedef struct _struct_NVDK_STRUCT_MULTICAST_HEADER
{
    unsigned char id;           /* Not used */
    unsigned char sub_id;       /* 0 -- video , 1 -- audio */
    unsigned char last;         /* 1: last packet of a frame. 0: otherwise */
    unsigned char packets;      /* This value is preserved, which is always 0*/
    unsigned char seq;          /* The sequence number in the fragmented UDP frames and
                                   started from 1 (1 ~ N), which will restart from 1 again
                                   when next frame arrived.  */
    unsigned char checksum;         /* This value is preserved, which is always 0*/
    unsigned char fpsmode_res; /* only for TCP1.0 where bit[7:4]:fps mode and
                                     bit[3:0] resolution index.
                                     for TCP2.0, this value is undefined */
    unsigned char fps_num;      /* only for TCP1.0 where bit[7:0]:fps number
```

```
                                    corresponding to the fps mode.
                                    for TCP2.0, this value is undefined. */
    unsigned int  frame_num;   /* frame counter, increased by 1. the video and audio
                                    has its own counter. */
    unsigned int  frame_len;   /* length of payload in a fragmented UDP packet. The
                                    Multicast Header is NOT included. */

}NVDK_STRUCT_MULTICAST_HEADER;
```

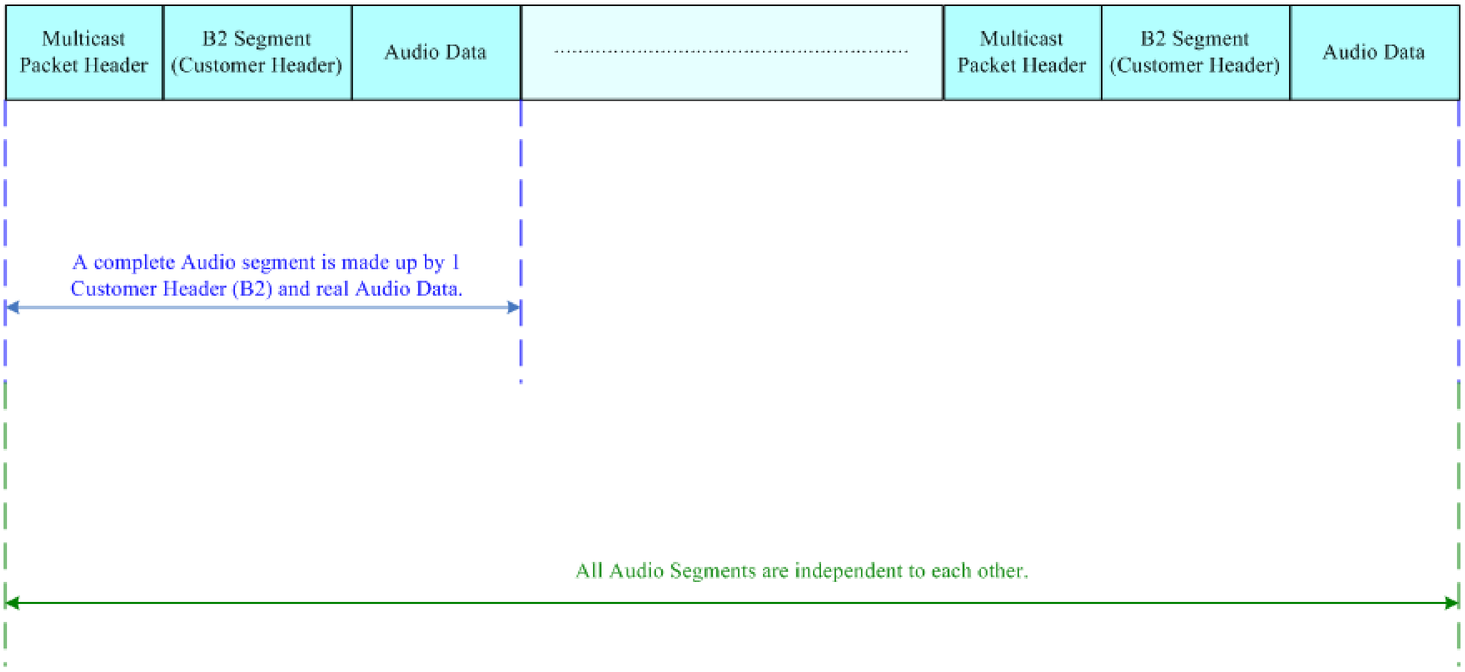
Important note:

1. Key packet attribute is very important to determine the last packet of the frame.

2. Need to find out Resolution and FPS from Sequence Header

3. 1(I or P frame) frame may divide into several multicast packets, each with a multicast packet header in front of it.

Multicast v2.0 Audio Packet Format


Multicast Packet Header
B2 Segment (Customer Header)
Audio Data
....
Multicast Packet Header
B2 Segment (Customer Header)
Audio Data
A complete Audio segment is made up by 1 Customer Header (B2) and real Audio Data.
All Audio Segments are independent to each other.

Note. Video server will send Audio & Video data in random order.

```
typedef struct tagMCPacketHead
{
unsigned char StreamId;
unsigned char StreamSubId;
unsigned char KeyPacket;
unsigned char TotalPacket;
unsigned char PacketNum;
unsigned char FrameCheckSum;
unsigned char Resolution;
unsigned char Fps;
unsigned int  FrameNum;
unsigned int  FrameLen;
} MCPacketHead;
```

RTP Packet Format

RTP over UDP :

Video :

(a) MPEG4 I Frame

RTP Header	Bitstream Data			I-Frame Data			VIDEO_B2_FRAME	
	Header	Data	Sequence	Header	Data	Frame data	B2_HEADER	
			0x00000100			N Byte	0x000001B2	User Data

(b) MPEG4 P Frame

RTP Header	P-Frame Data		VIDEO_B2_FRAME	
	Header	Frame data	B2_HEADER	
		N Byte	0x000001B2	User Data

Audio:

(c) Audio Frame

RTP Header	AUDIO_B2		Audio Data
	B2_HEADER		(PCM data)
	0x000001B2	User Data	N Byte

RTP over Multicast :

Video :

(a) I Frame

<table>
<tr><td rowspan="3">RTP Header</td><td colspan="3">Bitstream Data</td><td colspan="3">I-Frame Data</td><td colspan="2">VIDEO_B2_FRAME</td></tr>
<tr><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Sequence</td><td rowspan="2">Header</td><td rowspan="2">Data</td><td>Frame data</td><td>B2_HEADER</td><td></td></tr>
<tr><td>0x00000100</td><td>N Byte</td><td>0x000001B2</td><td>User Data</td></tr>
</table>

(b) P Frame

<table>
<tr><td rowspan="3">RTP Header</td><td colspan="2">P-Frame Data</td><td colspan="2">VIDEO_B2_FRAME</td></tr>
<tr><td rowspan="2">Header</td><td>Frame data</td><td>B2_HEADER</td><td></td></tr>
<tr><td>N Byte</td><td>0x000001B2</td><td>User Data</td></tr>
</table>

Audio:

(c) Audio Frame

<table>
<tr><td rowspan="3">RTP Header</td><td colspan="2">AUDIO_B2</td><td>Audio Data</td></tr>
<tr><td>B2_HEADER</td><td></td><td>(PCM data)</td></tr>
<tr><td>0x000001B2</td><td>User Data</td><td>N Byte</td></tr>
</table>

Note that RTP/RTSP protocol is implemented in TCP v2.0 compliant devices.

The details of RTP/RTSP protocol can be in RFC 2326 (RTSP) and RFC 3550 (RTP).

RTP Interface

SDP description :

```
v=0
o=- 1072886400760000 1 IN IP4 192.168.1.100
s=LIVE.COM Session streamed by a GO7007SB WISchip
i=LIVE.COM Streaming Media v
t=0 0
a=tool:LIVE.COM Streaming Media v2004.12.28
a=type:broadcast
a=control:*
a=range:npt=0-
a=x-qt-text-nam:LIVE.COM Session streamed by a GO7007SB WISchip
a=x-qt-text-inf:LIVE.COM Streaming Media v
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96
profile-level-id=245;config=000001B0F5000001B509000001000000012000C888
BAA760FA62D087828307
a=control:track1
m=audio 0 RTP/AVP 111
c=IN IP4 0.0.0.0
a=rtpmap:111 L16/8000
a=control:track2
```

RTSP request command :

[OPTIONS request]

```
rtsp://192.168.1.254:7070/ RTSP/1.0
CSeq: 1
User-Agent: VLC Media Player (LIVE.COM Streaming Media v2004.11.11)
```

[OPTIONS response]

```
sending response: RTSP/1.0 200 OK
CSeq: 1
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE
```

[DESCRIBE request]

```
DESCRIBE rtsp://192.168.1:100:7070 RTSP/1.0
CSeq: 1
Accept: application/sdp
Bandwidth: 384000
Accept-Language: en-GB
User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
```

[DESCRIBE response]

```
sending response: RTSP/1.0 200 OK
CSeq: 1
Date: Fri, Dec 02 2005 06:38:53 GMT
Content-Base: rtsp://192.168.1.100:7070//
Content-Type: application/sdp
Content-Length: 608

v=0
o=- 1133505497174429 1 IN IP4 192.168.1.100
s=LIVE.COM Session streamed by a GO7007SB WISchip
i=LIVE.COM Streaming Media v
t=0 0
a=tool:LIVE.COM Streaming Media v2004.12.28
a=type:broadcast
a=control:*
a=range:npt=0-
a=x-qt-text-nam:LIVE.COM Session streamed by a GO7007SB WISchip
a=x-qt-text-inf:LIVE.COM Streaming Media v
```

```
    m=video 0 RTP/AVP 96
    c=IN IP4 0.0.0.0
    a=rtpmap:96 MP4V-ES/90000
    a=fmtp:96 
profile-level-id=245;config=000001B0F5000001B50900000100000012000C888BAA760FA
62D087828307
    a=control:track1
    m=audio 0 RTP/AVP 111
    c=IN IP4 0.0.0.0
    a=rtpmap:111 L16/8000
    a=control:track2
```

[SETUP request]

```
    SETUP rtsp://192.168.1.100:7070//track1 RTSP/1.0
    CSeq: 2
    Transport: RTP/AVP;unicast;client_port=6970-6971
    x-retransmit: our-retransmit
    x-dynamic-rate: 1
    x-transport-options: late-tolerance=2.900000
    User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
    Accept-Language: en-GB
```

[SETUP response]

```
    sending response: RTSP/1.0 200 OK
    CSeq: 2
    Date: Fri, Dec 02 2005 06:38:54 GMT
    Transport: 
RTP/AVP;unicast;destination=192.168.1.3;client_port=6970-6971;server_port=1024
-1025
    Session: 1
```

[SETUP request]

```
    rtsp://192.168.1.100:7070//track2 RTSP/1.0
    CSeq: 3
    Transport: RTP/AVP;unicast;client_port=6972-6973
    x-retransmit: our-retransmit
    x-dynamic-rate: 1
    x-transport-options: late-tolerance=2.900000
    Session: 1
    User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
    Accept-Language: en-GB
```

[SETUP response]

```
    sending response: RTSP/1.0 200 OK
    CSeq: 3
    Date: Fri, Dec 02 2005 06:38:54 GMT
    Transport: 
RTP/AVP;unicast;destination=192.168.1.3;client_port=6972-6973;server_port=1026
-1027
    Session: 1
```

[PLAY request]

```
    rtsp://192.168.1.100:7070 RTSP/1.0
    CSeq: 4
    Range: npt=0.000000-
    x-prebuffer: maxtime=2.000000
    Session: 1
    User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
```

[PLAY response]

```
    sending response: RTSP/1.0 200 OK
    CSeq: 4
    Date: Fri, Dec 02 2005 06:38:54 GMT
    Range: npt=0.000-
    Session: 1
    RTP-Info: 
url=rtsp://192.168.1.100:7070//track1;seq=64955,url=rtsp://192.168.1.100:7070/
/track2;seq=39531
```

[PAUSE request]

```
    PAUSE rtsp://192.168.1.100:7070 RTSP/1.0
    CSeq: 5
    Session: 1
    User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
```

[PAUSE response]

```
sending response: RTSP/1.0 200 OK
CSeq: 5
Date: Fri, Dec 02 2005 06:39:36 GMT
Session: 1
```

[TEARDOWN request]

```
rtsp://192.168.1.100:7070 RTSP/1.0
CSeq: 6
Session: 1
User-Agent: QuickTime/7.0.3 (qtver=7.0.3;os=Windows NT 5.1Service Pack 1)
```

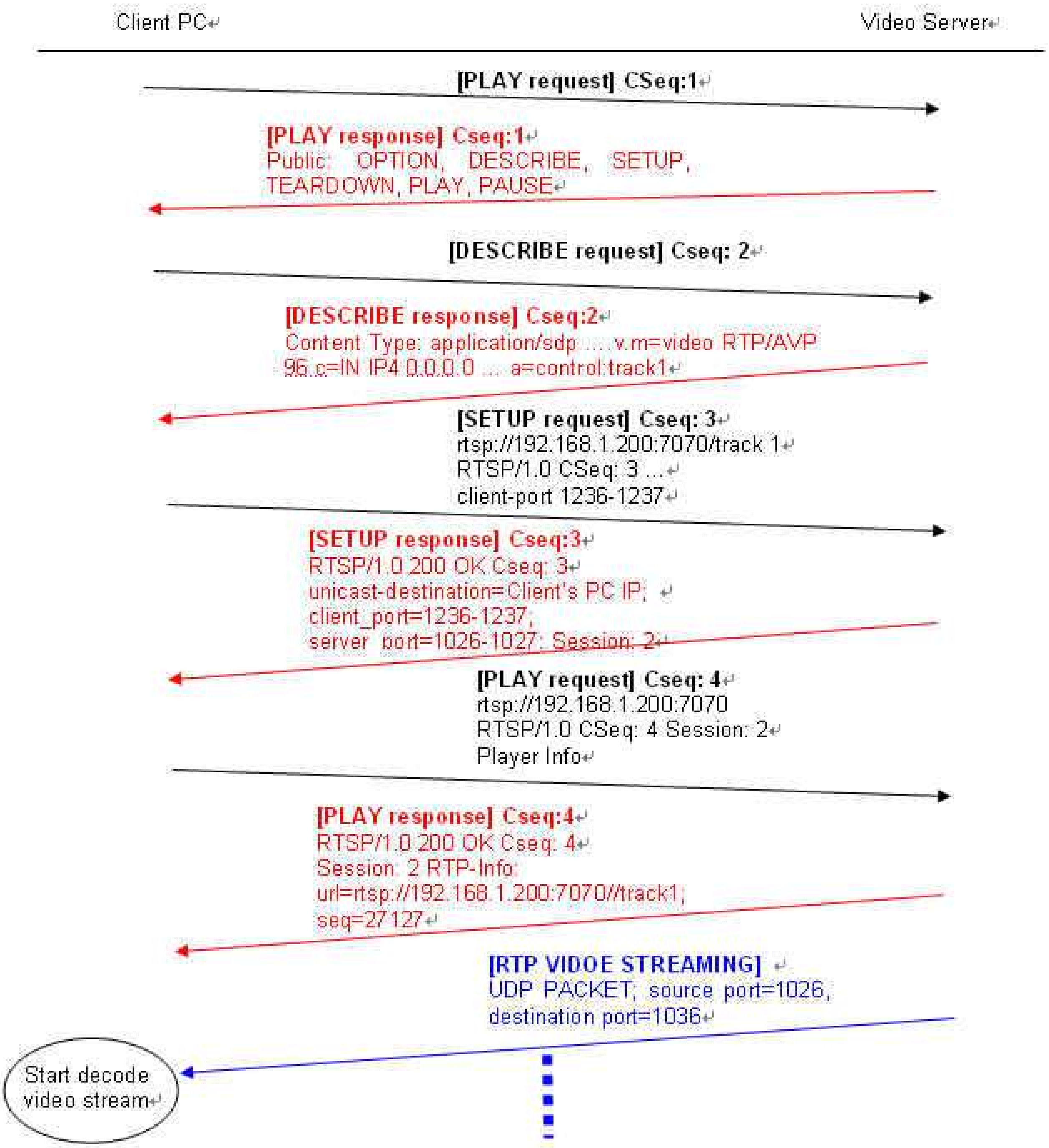
[TEARDOWN response]

```
sending response: RTSP/1.0 200 OK
CSeq: 6
Date: Fri, Dec 02 2005 06:39:36 GMT
```

Play an unicast RTP video stream (TRACK 1), while play an unicast audio stream (TRACK 2)

RTP Protocol Flow

Establishment



[PLAY request] CSeq:1

[PLAY response] Cseq:1
Public: OPTION, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE

[DESCRIBE request] Cseq: 2

[DESCRIBE response] Cseq:2
Content Type: application/sdp ….v.m=video RTP/AVP 96.c=IN IP4 0.0.0.0 … a=control:track1

[SETUP request] Cseq: 3
rtsp://192.168.1.200:7070/track 1
RTSP/1.0 CSeq: 3 …
client-port 1236-1237

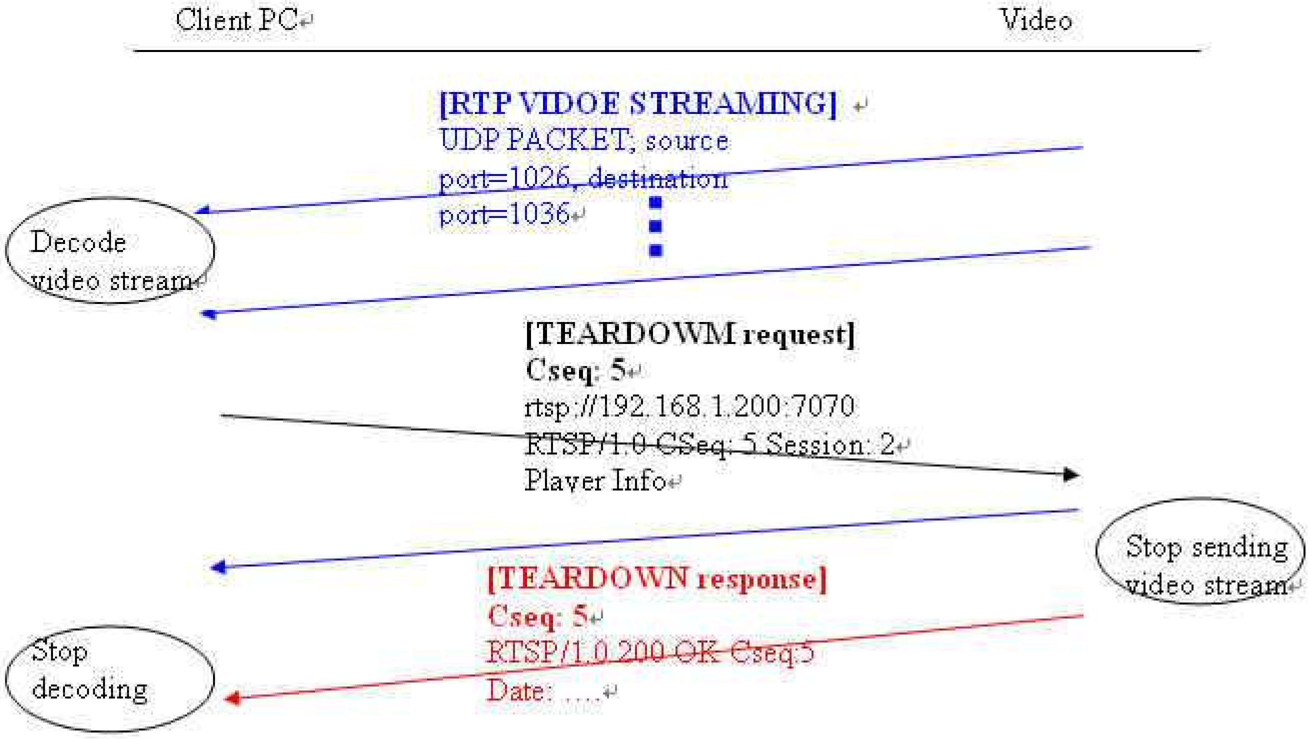
[SETUP response] Cseq:3
RTSP/1.0 200 OK Cseq: 3
unicast-destination=Client's PC IP;
client_port=1236-1237;
server_port=1026-1027; Session: 2

[PLAY request] Cseq: 4
rtsp://192.168.1.200:7070
RTSP/1.0 CSeq: 4 Session: 2
Player Info

[PLAY response] Cseq:4
RTSP/1.0 200 OK Cseq: 4
Session: 2 RTP-Info:
url=rtsp://192.168.1.200:7070//track1;
seq=27127

[RTP VIDOE STREAMING]
UDP PACKET; source port=1026,
destination port=1036

TEARDOWN An Unicast RTP VIDEO STREAM

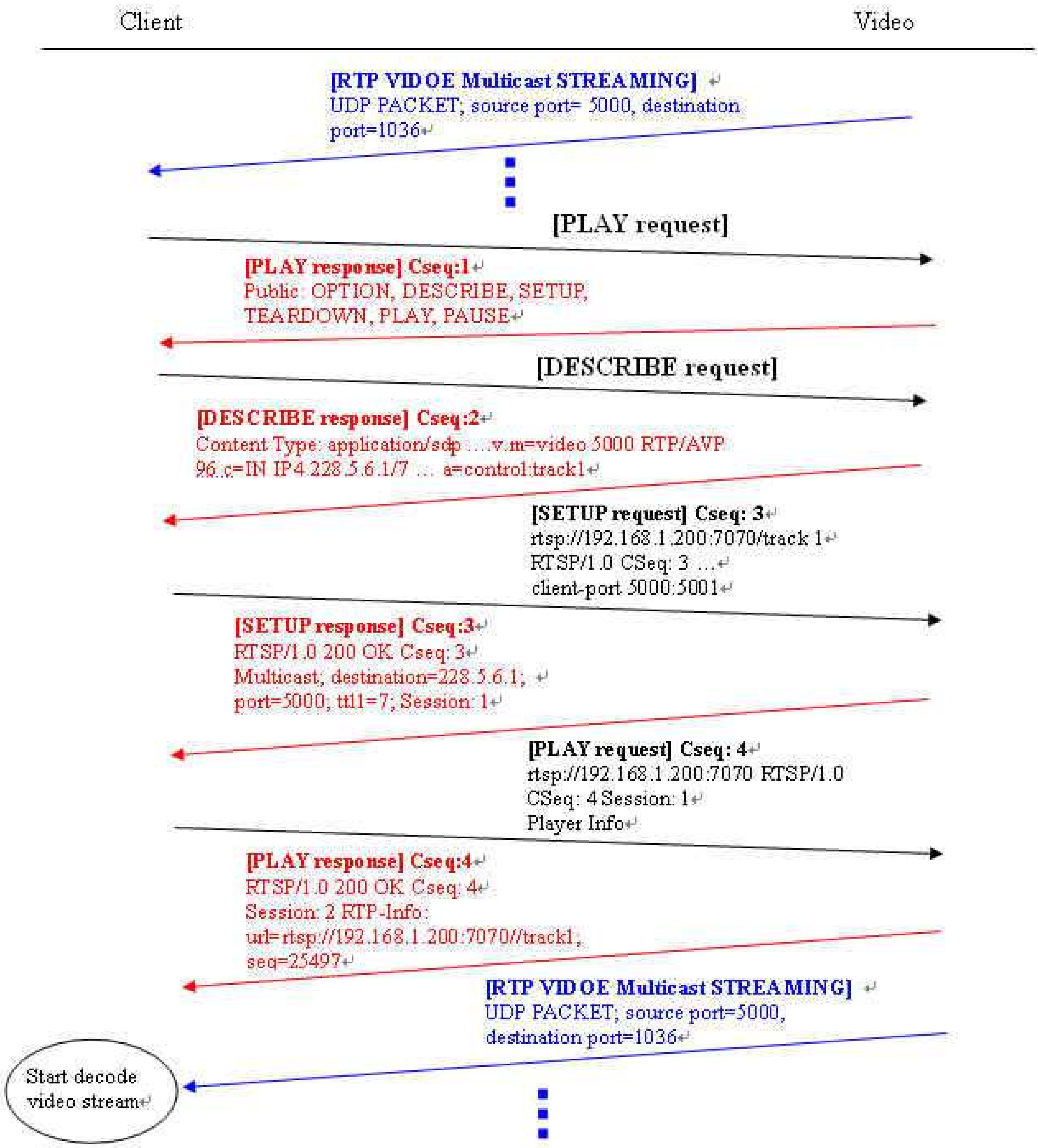


[RTP VIDOE STREAMING]
UDP PACKET; source port=1026, destination port=1036

[TEARDOWM request]
Cseq: 5
rtsp://192.168.1.200:7070
RTSP/1.0 CSeq: 5 Session: 2
Player Info

[TEARDOWN response]
Cseq: 5
RTSP/1.0 200 OK Cseq:5
Date: ….

PLAY A Multicast RTP VIDEO STREAM (TRACK 1):



[RTP VIDOE Multicast STREAMING]
UDP PACKET; source port= 5000, destination port=1036

[PLAY request]

[PLAY response] Cseq:1
Public: OPTION, DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE

[DESCRIBE request]

[DESCRIBE response] Cseq:2
Content Type: application/sdp ….v.m=video 5000 RTP/AVP 96..c=IN IP4 228.5.6.1/7 … a=control:track1

[SETUP request] Cseq: 3
rtsp://192.168.1.200:7070/track 1
RTSP/1.0 CSeq: 3 …
client-port 5000:5001

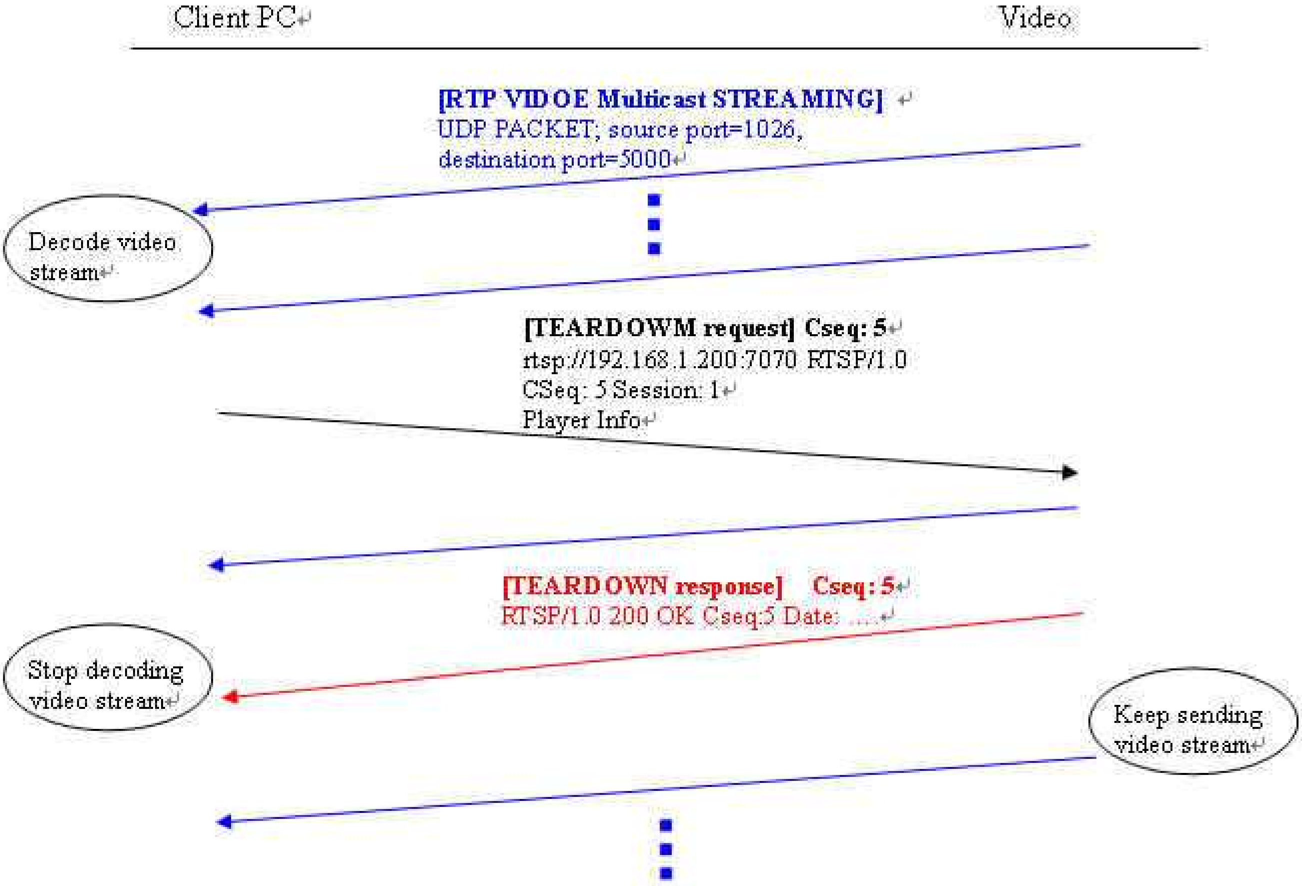
[SETUP response] Cseq:3
RTSP/1.0 200 OK Cseq: 3
Multicast; destination=228.5.6.1;
port=5000; ttl1=7; Session: 1

[PLAY request] Cseq: 4
rtsp://192.168.1.200:7070 RTSP/1.0
CSeq: 4 Session: 1
Player Info

[PLAY response] Cseq:4
RTSP/1.0 200 OK Cseq: 4
Session: 2 RTP-Info:
url=rtsp://192.168.1.200:7070//track1;
seq=25497

[RTP VIDOE Multicast STREAMING]
UDP PACKET; source port=5000, destination port=1036

TEARDOWN A Multicast RTP VIDEO STREAM


Client PC
Video
[RTP VIDOE Multicast STREAMING]
UDP PACKET; source port=1026,
destination port=5000
Decode video stream
[TEARDOWM request] Cseq: 5
rtsp://192.168.1.200:7070 RTSP/1.0
CSeq: 5 Session: 1
Player Info
[TEARDOWN response] Cseq: 5
RTSP/1.0 200 OK Cseq:5 Date: ....
Stop decoding video stream
Keep sending video stream

12

Migration Plan from SDK-2000 to SDK-10000

SDK-10000 New Features

SDK-10000 v1.0 series contains new design architecture with following features:

- Unified SDK for IP devices (IP Camera, Video Server, IP Speed Dome, and Quad Video Server), Capture Cards, Decoder Cards, Streaming Engine and File Playback. One programming can fit all above devices.
- Superset of SDK-2000, SDK-4000 and SDK-5000
- Scalable architecture: new adaptor can be added without changing codes
- Better performance: SDK-10000 has better performance and memory management over previous SDK. It also provide shorter video latency than previous SDK
- New adaptors: Direct Draw and FAVI (record to AVI file) adaptors provided
- Multi-channel Support: Supports multiple channel devices, 2/4/8-channel video server, 4-channel capture card

SDK-2000 vs SDK-10000 Function Calls

SDK – 2000	SDK – 10000	Remark
`netGetTCPMode`	`KGetTCPTypeByHTTP`	
`netOpenInterface`	`KOpenInterface`	
`netRegisterServer`	`KSetMediaConfig2` `KConnect`	
`netInitStream`	`KSetRenderInfo`	
`netStartStream`	`KStartStreaming` `KPlay`	

`netSetStatusCallBack`	`KSetVideoLossCallback2`
	`KSetVideoRecoveryCallback2`
	`KSetNetworkLossCallback`	

`netSetMDCallBack`	`KSetMotionDetectionCallback`	
`netSetDIDefault`	`KSetDIDefaultValue`	
`netSetDIOCallBack`	`KSetDICallback`	
`netSetTimeCodeCallBack`	`KSetTimeCodeCallback`	
`netSetAfterFlushCallBack`		Not support in SDK-10000
`netSetAfterRenderCallBack`	`KSetAfterRenderCallback`	
`netSetImageCallBack`	`KSetImageCallback`	
`netSetRS232CallBack`	`KSetRS232DataCallback`	
`netSetServerSerialDataCallBack`	`KSetRS232DataCallback`	
`netUnRegisterServer`	`KDisconnect`	
`netGetServerConfig`	`KGetVideoConfig2`	
`netSetServerConfig`	`KSetVideoConfig2`	
`netStopStream`	`KStopStreaming`	
`netSetAutoFrameRate`		Not support in SDK-10000
`netSetAlarmPreRecordingTime`	`KSetPrerecordTime`	
`netStartAlarmRecord`	`KStartRecord`	

netStopAlarmRecord	KStopRecord	
netStopAlarmRecord2	KStopRecord	
netStartRecord	KStartRecord	
netStopRecord	KStopRecord	
netStopRecord2	KStopRecord	
netSend2ServerSerialPort	KSendRS232Command	
netSendKeyPadCommand	KSendPTZCommand	
netSendDIO	KSendDO	
netSetMotionRange	KSetMotionInfo	
netSetMotionSensitive	KGetMotionInfo KSetMotionInfo	
netGetLastError	KGetLastError	
netGetFrameReceived	KGetTotalReceiveVideoFrameCount	
netGetDataReceived	KGetTotalReceiveSize	
netGetDispWindowPos		Not support in SDK-10000
netSetDispWindowPos	KSetRenderInfo	
netSetRS232	KSendRS232Setting	
netSetServerSerialPort	KSendRS232Setting	
netSearchServer		Sample/SearchSample
netGetDioStatus	KGetDIDefaultValueByHTTP	
netGetMotionSetting	KGetMotionInfo	
netSetMpeg4RawCallBack	KSetRawDataCallback	
netGetOnLineUser	KGetOnLineUser	
netGetSDKVersion	KGetVersion	
netGetServerVersion	KGetServerVersion	
netRegisterServerEx	KSetMediaConfig2 KConnect	
netSetCommunicationPort		Not support in SDK-10000
netDecodeI	KSetDecodeIFrameOnly	
netSendURL	KSendURLCommand	

netSendCMD		Not support in SDK-10000
netCloseInterface	KCloseInterface	
netGetCameraName	KGetCameraName	
netSaveReBoot	KSaveReboot	
netGetControlToken		Not support in SDK-10000
netGetAudioToken	KGetAudioToken	
netFreeControlToken		Not support in SDK-10000
netGiveOffSound		Not support in SDK-10000
netCloseSound	KStopAudioTransfer	
netFreeAudioToken	KFreeAudioToken	
netIsMute		Not support in SDK-10000
netSetVolume	KSetVolume	
netGetVolume	KGetVolume	
netSetPreviewBuffer		Not support in SDK-10000
netSendAudio	KStartAudioTransfer	
netSetMpeg4RawCallBack2	KSetRawDataCallback	
netSetAudioRawCallBack	KSetRawDataCallback	
netSetStreamRawCallBack	KSetRawDataCallback	
netSend2StreamEngine	KSendCommandToStreamingEngine	
netMute	KSetMute	
netSetSocketSize		Not support in SDK-10000
netRegisterServerControlOnly	KSetMediaConfig2 KConnect	Set Contact type to CONTACT_TYPE_CONTROL_ONLY
netStartWriteInfo		Not support in SDK-10000
netStopWriteInfo		Not support in SDK-10000
netGetDeviceType	KGetDeviceTypeByHTTP	
netSetHTTPPort		Not support in SDK-10000
netSetResolutionChangeCallBack	KSetResolutionChangeCallback	
netSetChannelNumber		Not support in SDK-10000

`netSetConnectTimeOut`		Not support in SDK-10000

Application Migration Guide

This section describes the steps for customers to port their application from SDK-2000 to SDK-10000.

We provide 2 different step-by-step guides for following applications:

- Application that uses MPEG-4 raw data only
- Application that uses most function calls

Application that uses MPEG-4 raw data only

Steps to migrate from SDK-2000 to SDK-10000:

1. Re-compile the source codes with SDK-10000
2. Use `KGetTCPTypeByHTTP()` to detect if the device is compatible to TCP 1.0 format or TCP 2.0 (supports audio) format
3. Use `KSetRawDataCallback()` to receive both Raw-Video and Raw-Audio data.
4. Use `KSetImageCallback()` to get RGB buffer at the same time
5. Call `KSendPTZCommand()` to send PTZ commands.
6. Note that in SDK-10000, every I-Frame contains sequence header in the frame
7. Refer to Audio API for 1-way or 2-way audio functions
8. Refer to MPEG-4 data structure section for detailed MPEG-4 audio + video format

Application that uses most function calls

Steps to migrate from SDK-2000 to SDK-10000:

1. Re-compile the source codes with SDK-10000
2. Use `KGetTCPTypeByHTTP()` to detect if the device is compatible to TCP 1.0 format or TCP 2.0 (supports audio) format
3. Use `KSetDecodeIFrameOnly()` function to decode I-Frame only to save CPU utilization; this will only affect on the decoding part, recording can still record with specified frame rate
4. Call `KSendPTZCommand()` to send PTZ commands.
5. Refer to Audio API for 1-way or 2-way audio functions.

A

APPANDIX A

How to custom a PTZ file

4.3.2008

Here we discuss how to custom a PTZ file for a camera device, and only those “important” commands are listed bellow. If you need some “minor” commands which are not described here, Please refer to the PTZ protocol files in SDK folder.

1. Attributes:

Example:
[ATTRIBUTES]
ADDRIDSTART; 0x01
ADDRIDPOS; 1
CHECKSUM; $B6=$B1^$B2^$B3^$B4^$B5
PANEL; PANTILT,MOVE,ZOOM,FOCUS,IRIS,BLC,PRESET

Description:
The “ADDRIDSTART” indicate the address ID in BYTE.
The “ADDRIDPOS” is address ID starting position. “1” is the first byte in PTZ command
The “CHECKSUM” is “HOW to calculate the checksum” and “WHERE to place the checksum”.
$B6 indicates 6th position in PTZ command.
$C5 indicates a constant ‘5’.
The defined operator is ‘+’ ‘-’ ‘*’ ‘/’ ‘^’ ‘|’ ‘&’.

If the “$D” is found in checksum string, that means no checksum should be calculated.

The “PANEL” indicates which panel you need.

2. PANTILT

Example:

```
[PANTILT]
PANLEFT;       1; 0;0x01,0x00,0x18,0x01,0x01,0x18
PANLEFT;       2; 0;0x01,0x00,0x18,0x01,0x02,0x1C
PANLEFT;       3; 0;0x01,0x00,0x18,0x01,0x03,0x10
PANLEFT;       4; 0;0x01,0x00,0x18,0x01,0x05,0x14
PANLEFT;       5; 0;0x01,0x00,0x18,0x01,0x07,0x17
PANLEFT;       0; 0;0x01,0x00,0x13,0x00,0x00,0x12

PANRIGHT;      1; 0;0x01,0x00,0x18,0x00,0x01,0x19
PANRIGHT;      2; 0;0x01,0x00,0x18,0x00,0x02,0x1D
PANRIGHT;      3; 0;0x01,0x00,0x18,0x00,0x03,0x11
PANRIGHT;      4; 0;0x01,0x00,0x18,0x00,0x05,0x15
PANRIGHT;      5; 0;0x01,0x00,0x18,0x00,0x07,0x16
PANRIGHT;      0; 0;0x01,0x00,0x13,0x00,0x00,0x12

TILTUP;        1; 0;0x01,0x00,0x18,0x02,0x01,0x1B
TILTUP;        2; 0;0x01,0x00,0x18,0x02,0x02,0x1F
TILTUP;        3; 0;0x01,0x00,0x18,0x02,0x03,0x13
TILTUP;        4; 0;0x01,0x00,0x18,0x02,0x05,0x17
TILTUP;        5; 0;0x01,0x00,0x18,0x02,0x07,0x14
TILTUP;        0; 0;0x01,0x00,0x14,0x00,0x00,0x15

TILTDOWN;      1; 0;0x01,0x00,0x18,0x03,0x01,0x1A
TILTDOWN;      2; 0;0x01,0x00,0x18,0x03,0x02,0x1E
TILTDOWN;      3; 0;0x01,0x00,0x18,0x03,0x03,0x12
TILTDOWN;      4; 0;0x01,0x00,0x18,0x03,0x05,0x16
TILTDOWN;      5; 0;0x01,0x00,0x18,0x03,0x07,0x15
TILTDOWN;      0; 0;0x01,0x00,0x14,0x00,0x00,0x15

PANTILTSTOP; 0; 0;0x81,0x01,0x06,0x01,0x00,0x00,0x03,0x03,0xFF
```

Description:

The “PANLEFT”, “PANRIGHT”, “TILTUP”, and “TILTDOWN” commands should be described here. The first parameter is the “speed”, the second parameter is reserved (0). If there is no “PANTILTSTOP” command, speed 0 must be there instead.

3. ZOOM

Example:

```
[ZOOM]
ZOOMIN;      1; 0;0x01,0x00,0x24,0x01,0x00,0x24
#ZOOMIN;      0; 0;
ZOOMOUT;    1; 0;0x01,0x00,0x24,0x00,0x00,0x25
#ZOOMOUT;    0; 0;
ZOOMSTOP; 0; 0;0x01,0x00,0x24,0x04,0x00,0x21
```

Description:

The “ZOOMIN”, “ZOOMOUT”, and “ZOOMSTOP” commands should be described here. The first parameter is 1 for “ZOOMIN” and “ZOOMOUT”, 0 for “ZOOMSTOP”.

BTW: The ‘#’ mark the line disabled.

4. FOUCUS

Example:

```
[FOCUS]
FOCUSIN; 1; 0;0x81,0x01,0x04,0x08,0x03,0xFF
FOCUSOUT; 1; 0;0x81,0x01,0x04,0x08,0x02,0xFF
FOCUSSTOP; 0; 0;0x81,0x01,0x04,0x08,0x00,0xFF
```

Description:

The “FOCUSIN”, “FOCUSOUT”, and “FOCUSSTOP” commands should be described here. The first parameter is 1 for “FOCUSIN” and “FOCUSOUT”, 0 for “FOCUSSTOP”.

5. IRIS

Example:

```
[IRIS]
IRISOPEN;1;0;0x81,0x01,0x04,0x0B,0x02,0xFF
```

```
IRISSTOP;0;0;0x81,0x01,0x04,0x0B,0x00,0xFF
IRISCLOSE;1;0;0x81,0x01,0x04,0x0B,0x03,0xFF
```

Description:

The “IRISOPEN”, “IRISCLOSE”, and “IRISSTOP” commands should be described here. The first parameter is 1 for “IRISOPEN” and “IRISCLOSE”, 0 for “IRISSTOP”.

6. PRESET

Example:

```
[PRESET]
PRESETGOTO;1;0;0x81,0x01,0x04,0x3F,0x02,0x00,0xFF
PRESETGOTO;2;0;0x81,0x01,0x04,0x3F,0x02,0x01,0xFF
PRESETGOTO;3;0;0x81,0x01,0x04,0x3F,0x02,0x02,0xFF
PRESETGOTO;4;0;0x81,0x01,0x04,0x3F,0x02,0x03,0xFF
PRESETGOTO;5;0;0x81,0x01,0x04,0x3F,0x02,0x04,0xFF
PRESETGOTO;6;0;0x81,0x01,0x04,0x3F,0x02,0x05,0xFF
PRESETSET;1;0;0x81,0x01,0x04,0x3F,0x01,0x00,0xFF
PRESETSET;2;0;0x81,0x01,0x04,0x3F,0x01,0x01,0xFF
PRESETSET;3;0;0x81,0x01,0x04,0x3F,0x01,0x02,0xFF
PRESETSET;4;0;0x81,0x01,0x04,0x3F,0x01,0x03,0xFF
PRESETSET;5;0;0x81,0x01,0x04,0x3F,0x01,0x04,0xFF
PRESETSET;6;0;0x81,0x01,0x04,0x3F,0x01,0x05,0xFF
PRESETCLEAR;1;0;0x81,0x01,0x04,0x3F,0x00,0x00,0xFF
PRESETCLEAR;2;0;0x81,0x01,0x04,0x3F,0x00,0x01,0xFF
PRESETCLEAR;3;0;0x81,0x01,0x04,0x3F,0x00,0x02,0xFF
PRESETCLEAR;4;0;0x81,0x01,0x04,0x3F,0x00,0x03,0xFF
PRESETCLEAR;5;0;0x81,0x01,0x04,0x3F,0x00,0x04,0xFF
PRESETCLEAR;6;0;0x81,0x01,0x04,0x3F,0x00,0x05,0xFF

PRESETTOUR;1;0;0x01,0x00,0x11,0x01,0x00,0x11
```

Description:

The “PRESETCLEAR”, “PRESETGOTO” and “PRESETSET” define preset positions. The “PRESETSET” is adding a preset position. The “PRESETCLEAR” is removing a preset position. The “PRESETGOTO” make camera moving to one preset position. The first parameter is the ID of a preset position, second parameter is reserved (0). If there is a command to make camera touring every preset position, describe it behind “PRESETTOUR”.

7. OSD

Example:

```
[OSD]
OSDON;        0; 0;0x01,0x00,0x28,0x04,0x00,0x2D
#OSDOFF;      0; 0;0x01,0x00,0x28,0xFF,0x00,0xD6
OSDUP;        0; 0;0x01,0x00,0x28,0x00,0x00,0x29
OSDDOWN;    0; 0;0x01,0x00,0x28,0x01,0x00,0x28
OSDLEFT;    0; 0;0x01,0x00,0x28,0x02,0x00,0x2B
OSDRIGHT; 0; 0;0x01,0x00,0x28,0x03,0x00,0x2A
OSDENTER; 0; 0;0x01,0x00,0x28,0x04,0x00,0x2D
OSDLEAVE; 0; 0;0x01,0x00,0x28,0xFF,0x00,0xD6
#OSDSTOP; 0; 0;0xA0,0x00,0x00,0x00,0x00,0x00,0xAF,0x00
```

Description:

The [OSD] section defined how to operate OSD functions. There are 9 definitions for OSD actions.

B

APPENDIX B

Authentication Sample in RTP/RTSP

Notify: 1.The EOL of SDP header is \r\n
2.Every EOL of content is possible \r\n or \n. (Total length match with content-length)

```
##################################################################################
1. Digest Algorithm Authentication in RTSP
   RTP Over UDP
##################################################################################
OPTIONS rtsp://Admin:123456@172.16.3.62:7070 RTSP/1.0
CSeq: 1
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)



RTSP/1.0 200 OK
CSeq: 1
Date: Thu, Jan 01 2004 00:02:11 GMT
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY



DESCRIBE rtsp://Admin:123456@172.16.3.62:7070 RTSP/1.0
CSeq: 2
Accept: application/sdp
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)



RTSP/1.0 401 Unauthorized
CSeq: 2
Date: Thu, Jan 01 2004 00:02:11 GMT
WWW-Authenticate: Digest realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5"
```

```
DESCRIBE rtsp://Admin:123456@172.16.3.62:7070 RTSP/1.0
CSeq: 3
Accept: application/sdp
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5",
uri="rtsp://Admin:123456@172.16.3.62:7070",
response="b0bf040dce84753d2e1e11c505b11a88"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)


RTSP/1.0 200 OK
CSeq: 3
Date: Thu, Jan 01 2004 00:02:11 GMT
Content-Base: rtsp://Admin:123456@172.16.3.62:7070/
Content-Type: application/sdp
Content-Length: 563

v=0
o=- 107291533100310000 1 IN IP4 172.16.3.62
s=Session streamed by RTP/RTSP server
i=COM Streaming Media v
t=0 0
a=tool:COM Streaming Media v2006.10.22
a=type:broadcast
a=control:*
a=range:ntp=0-
a=x-qt-text-name:Session streamed by RTP/RTSP server
a=x-qt-text-inf:COM Streaming Media v
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=245;config=000001B0F5000001B509000001000000012000006
a=control:track1
m=audio 0 RTP/AVP 111
c=IN IP4 0.0.0.0
a=rtpmap:111 L16/8000
```

```
a=control:track2

SETUP rtsp://Admin:123456@172.16.3.62:7070/track1 RTSP/1.0
CSeq: 4
Transport: RTP/AVP;unicast;client_port=1808-1809
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5",
uri="rtsp://Admin:123456@172.16.3.62:7070",
response="8b96c1f004f68f34e27003ff628eee58"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)


RTSP/1.0 200 OK
CSeq: 4
Date: Thu, Jan 01 2004 00:02:11 GMT
Transport:
RTP/AVP;unicast;destination=172.16.3.45;client_port=1808-1809;server_port=1000-100
1
Session: 1


SETUP rtsp://Admin:123456@172.16.3.62:7070/track2 RTSP/1.0
CSeq: 5
Transport: RTP/AVP;unicast;client_port=1810-1811
Session: 1
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5",
uri="rtsp://Admin:123456@172.16.3.62:7070",
response="8b96c1f004f68f34e27003ff628eee58"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)


RTSP/1.0 200 OK
CSeq: 5
Date: Thu, Jan 01 2004 00:02:11 GMT
Transport:
```

```
RTP/AVP;unicast;destination=172.16.3.45;client_port=1810-1811;server_port=1002-1003
Session: 1


PLAY rtsp://Admin:123456@172.16.3.62:7070 RTSP/1.0
CSeq: 6
Session: 1
Range: npt=0.000-
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5",
uri="rtsp://Admin:123456@172.16.3.62:7070",
response="d2afe212004430fb0ef30db737423444"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)


RTSP/1.0 200 OK
CSeq: 6
Date: Thu, Jan 01 2004 00:02:11 GMT
Range: npt=0.000-
Session: 1
RTP-Info:
url=rtsp://172.16.3.45:7070//track1;seq=7793,url=rtsp://172.16.3.45:7070//track2;seq=5386


TEARDOWN rtsp://Admin:123456@172.16.3.62:7070 RTSP/1.0
CSeq: 7
Session: 1
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="c25d8ee72e9e3ff654d9de6bb9f3efd5",
uri="rtsp://Admin:123456@172.16.3.62:7070",
response="018ee0e5539088e0218aee0cb1556283"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)
```

```
RTSP/1.0 200 OK
CSeq: 7
Date: Thu, Jan 01 2004 00:02:16 GMT


##################################################################################
1. Digest Algorithm Authentication in RTSP
   RTP Over Multicast
##################################################################################
OPTIONS rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 1
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
CSeq: 1
Date: Thu, Jan 01 2004 00:37:39 GMT
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY

DESCRIBE rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 2
Accept: application/sdp
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 401 Unauthorized
CSeq: 2
Date: Thu, Jan 01 2004 00:37:39 GMT
WWW-Authenticate: Digest realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e"

DESCRIBE rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 3
Accept: application/sdp
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e",
uri="rtsp://Admin:123456@172.16.3.14:7070",
response="790baa56c992fdab991160da96a75445"
```

```
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
CSeq: 3
Date: Thu, Jan 01 2004 00:37:39 GMT
Content-Base: rtsp://Admin:123456@172.16.3.14:7070/
Content-Type: application/sdp
Content-Length: 573

v=0
o=- 107291745900160000 1 IN IP4 172.16.3.14
s=Session streamed by RTP/RTSP server
i=COM Streaming Media v
t=0 0
a=tool:COM Streaming Media v2006.10.22
a=type:broadcast
a=control:*
a=range:ntp=0-
a=x-qt-text-name:Session streamed by RTP/RTSP server
a=x-qt-text-inf:COM Streaming Media v
m=video 5000 RTP/AVP 96
c=IN IP4 228.5.6.1
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=245;config=000001B0F5000001B50900000100000001200006
a=control:track1
m=audio 5002 RTP/AVP 111
c=IN IP4 228.5.6.1
a=rtpmap:111 L16/8000
a=control:track2

SETUP rtsp://Admin:123456@172.16.3.14:7070/track1 RTSP/1.0
CSeq: 4
Transport: RTP/AVP;multicast;client_port=5000-5001
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e",
uri="rtsp://Admin:123456@172.16.3.14:7070",
response="387a360cc9b70fc8c17f6e74bbfacd70"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
CSeq: 4
Date: Thu, Jan 01 2004 00:37:39 GMT
Transport: RTP/AVP;multicast;destination=228.5.6.1;port=5000;ttl=255
Session: 1

SETUP rtsp://Admin:123456@172.16.3.14:7070/track2 RTSP/1.0
CSeq: 5
Transport: RTP/AVP;multicast;client_port=5002-5003
Session: 1
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e",
uri="rtsp://Admin:123456@172.16.3.14:7070",
response="387a360cc9b70fc8c17f6e74bbfacd70"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
CSeq: 5
Date: Thu, Jan 01 2004 00:37:39 GMT
Transport: RTP/AVP;multicast;destination=228.5.6.1;port=5002;ttl=255
Session: 1

PLAY rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 6
Session: 1
Range: npt=0.000-
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e",
uri="rtsp://Admin:123456@172.16.3.14:7070",
response="2aee9818580c78f0f53e3ee4ce2f6f71"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
```

```
CSeq: 6
Date: Thu, Jan 01 2004 00:37:39 GMT
Range: npt=0.000-
Session: 1
RTP-Info:
url=rtsp://172.16.3.45:7070//track1;seq=14339,url=rtsp://172.16.3.45:7070//track2;
seq=6211

TEARDOWN rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 7
Session: 1
Authorization: Digest username="Admin", realm="Session streamed by RTP/RTSP server",
nonce="5269eb060e0385a667bb96c3562c542e",
uri="rtsp://Admin:123456@172.16.3.14:7070",
response="2ca566fc976ba0ef051ea5e215efafbb"
User-Agent: VLC media player (LIVE555 Streaming Media v2006.03.16)

RTSP/1.0 200 OK
CSeq: 7
Date: Thu, Jan 01 2004 00:37:44 GMT

##################################################################################
3. Base64 Algorithm Authentication in RTSP
   RTP Over UDP
##################################################################################

DESCRIBE rtsp://172.16.3.14:7070/udp/track1 RTSP/1.0
CSeq: 10
Authorization: Basic QWRtaW46MTIzNDU2

RTSP/1.0 200 OK
CSeq: 10
Date: Thu, Jan 01 2004 00:28:38 GMT
Content-Base: /
Content-Type: application/sdp
Content-Length: 483
```

```
v=0
o=- 107291691800790000 1 IN IP4 192.168.0.100
s=Session streamed by RTP/RTSP server
i=COM Streaming Media v
t=0 0
a=tool:COM Streaming Media v2006.10.22
a=type:broadcast
a=control:*
a=range:ntp=0-
a=x-qt-text-name:Session streamed by RTP/RTSP server
a=x-qt-text-inf:COM Streaming Media v
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=245;config=000001B0F5000001B50900000100000001200006
a=control:track1

SETUP rtsp://172.16.3.14:7070/udp/track1/track1 RTSP/1.0
CSeq: 11
Transport: RTP/AVP;unicast;client_port=15000-15001

RTSP/1.0 200 OK
CSeq: 11
Date: Thu, Jan 01 2004 00:28:38 GMT
Transport:
RTP/AVP;unicast;destination=172.16.3.79;client_port=15000-15001;server_port=1006-1
007
Session: 4

PLAY rtsp://172.16.3.14:7070/udp/track1 RTSP/1.0
CSeq: 12
Session: 4
Range: npt=0.000-

RTSP/1.0 200 OK
```

```
CSeq: 12
Date: Thu, Jan 01 2004 00:28:38 GMT
Range: npt=0.000-
Session: 4
RTP-Info: url=rtsp://172.16.3.79:7070//track1;seq=20059

TEARDOWN rtsp://Admin:123456@172.16.3.14:7070 RTSP/1.0
CSeq: 13
Session: 4
Authorization: Basic QWRtaW46MTIzNDU2


RTSP/1.0 200 OK
CSeq: 13
Date: Thu, Jan 01 2004 00:08:54 GMT
```