

AVN2XX Control Protocol (API)

Version E – released 02/18/2011

All control communication with the AVN2XX Audio/Video Encoder Box is done via UDP datagram socket communication. The AVN2XX unit will listen on UDP Port 7000 for incoming command requests. After constructing/binding/connecting a UDP datagram socket from your application to the AVN2XX unit, there are several control structures that can be sent across. I have included these structures in Appendix A. Each Structure begins with a common 4 byte header of which the first byte is used to specify the OpCode for a specific command. The trailing data in each structure is unique to the specific command. The AVN2XX unit will parse the incoming data and pass it on to the appropriate function based on the OpCode. After performing the desired action, the AVN2XX unit will send a response packet back to the client of the same size and structure of the original request. The first byte of the response will again contain the OpCode as it was sent across by the client, and in addition the AVN2XX unit will turn on either the Highest Order Bit (ERROR) or the Second Highest Order Bit (SUCCESS) so the result of the action can be interpreted by the client with a simple check like the following:

```
If (first_byte & 0x80)
    ERROR
Else if (first_byte & 0x40)
    SUCCESS
```

The same check can be used with every command because the original value (OpCode) will still be present, in addition to one of the 2 signal bits. A side effect of this current implementation is that we can only use the lower 6 bits for command OpCodes which limits us to 64 possible commands, but this limitation will be removed in the next Major Version Release of the code. A list of the current OpCodes is included in Appendix B

NOTE: The Control Protocol (API) is currently being reviewed, and some modifications of the structure formats, and sending/receiving formats will be modified in the near future. Please use these current instructions as a guideline, but do not take too much effort to Hardcode these settings into your application and/or test programs, as we expect some major changes soon. We will keep you posted when the changes do occur.

1.1.1.1.1.1 CMD_QUERY_VIDEO

Check current streaming status – (SessionQueryData – see Appendix A)

OPCODE = 0x11

3 bytes (0,0,0)

64 bytes (SessionSettings struct to be filled in)

NOTE: the second byte will contain product code:

1=AVN200, 3=AVN210, 5=AVN220 ...

NOTE: the fourth byte will contain encoding status:

0 = not encoding or no system integrity task running

1 = encoding/stream OK

2 = error in Analog Front End

3 = error in Encoding Device

1.1.1.1.1.2 CMD_START_VIDEO

Start AVN2XX stream – (SessionStartData struct – see Appendix A)

Byte #1 OPCODE = 0x12

Byte #2 0

Byte #3 0

Byte #4 0

32 bytes (username, ASCII)

32 bytes (password, ASCII)

64 bytes (SessionSettings struct)

{option byte #2, 0x45=Use Stored AVN Settings to Start}

{option byte #2, 0x46=Update Stored Settings, but do not Start}

{option byte #2, 0x34=Toggle Video Input; if (byte#4!=0) select byte#3, else toggle}

1.1.1.1.1.3 CMD_STOP_VIDEO

Stop AVN2XX Stream – (SessionStopData struct – see Appendix A)

OPCODE = 0x13

3 bytes (0,0,0)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

1.1.1.1.1.4 CMD_VERSION_VIDEO (DEV INFO)

Check AVN2XX version – (SessionVersionData struct – see Appendix A)

OPCODE = 0x23

3 bytes (0,0,0)

80 bytes (ASCII string, empty when passed, filled in on return)

OPTIONS:

3 bytes (0x23,0,0) Get Slot ID (AVN220 Chassis Location, >= 1_96)

Result returned in second byte (overwrites 0x23)

OR

3 bytes (0x24,0,0) Get Thermal Sensor reading (AVN220 only >= 1_96)

Result returned in bytes 3 and 4 (9 bits, 8 from B4 and 1 from B3)

OR
3 bytes (0x34,0,0) Get Boot Code Version (>=AVN1_96)
Result returned in same place a normal Version String, no additional parsing required
OR
3 bytes (0x22,0,0) Get Flash Device Type (>=1_98)
Result returned in byte #3, iff byte #4 is set to 0x22. (Otherwise FW too old)
OR
3 bytes (0x25,0,0) Get Unit Name (>=1_98)
Result returned in version string, iff Byte #2 is set to 0x25. (Otherwise FW too old)
OR
3 bytes (0x26,0,0) Get Unit Location (>=1_98)
Result returned in version string, iff Byte #2 is set to 0x26. (Otherwise FW too old)
OR
3 bytes (0x27,0,0) Get Unit Comments/Notes (>=1_98)
Result returned in version string, iff Byte #2 is set to 0x27. (Otherwise FW too old)
OR
3 bytes (0x28,0,0) Get Unit uptime in seconds (>=2_02)
Result returned as string in place of version string
OR
3 bytes (0x29,0,0) Get Unit cpu utilization (>=2_02)
Result returned as string in place of version string

1.1.1.1.1.5 CMD_RESET (SAVE / RESET)

Reset the AVN2XX unit – (SessionResetData struct – see Appendix A)
Byte #1 OPCODE = 0x07
Byte #2 SAVE: (1=video,2=options,3=flash params,4=audio,99=all,0=none)
Byte #3 REBOOT: (0=No, 1=Yes)
Same as CMD_VIDEO_STOP

1.1.1.1.1.6 CMD_STOP_LOCK

Set Stop lock mode – (use SessionStopData struct – see Appendix A)
OPCODE = 0x24
Same as CMD_VIDEO_STOP

1.1.1.1.1.7 CMD_QUERY_LOCK

Check current lock mode – 8 bytes
OPCODE = 0x25
3 bytes (0,0,0)
1 byte (sent as 0, result stored here) 0=unlocked, 1 = locked
3 bytes (0,0,0)

1.1.1.1.1.8 CMD_QUERY_AUTH

Check auth parameters – (use SessionStopData struct – see Appendix A)
OPCODE = 0x26
Same as CMD_VIDEO_STOP

1.1.1.1.1.9 CMD_SET_CAM_TYPE

Set Camera Type – (SessionCamTypeData struct – see Appendix A)

OPCODE = 0x27

3 bytes (0,0,0)

1 byte (port, 0=RS422, 1=RS232)

32 bits (baudrate in bps)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

1.1.1.1.1.10 CMD_QUERY_DIG_IN

Check Digital Input Values – (SessionDigIOData struct – see Appendix A)

OPCODE = 0x28

3 bytes (0,0,0)

3 bytes (sent as 0, filled in on return. 0=off, 1=on) (REV_C and greater uses only first 2)

1 byte (0) ignored on send, filled in with DOUT value (1=on, 0=off) on return

32 bytes (username, ASCII)

32 bytes (password, ASCII)

1.1.1.1.1.11 CMD_SET_DIG_OUT –

Set Digital Output Value – (SessionDigIOData struct – see Appendix A)

OPCODE = 0x29

3 bytes (0,0,0)

3 bytes (0,0,0) not used

1 byte (0 = off, 1 = on)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

1.1.1.1.1.12 CMD_GET_IP_CONFIG

Check IP configuration – (SessionConfigIPAll struct – see Appendix A)

OPCODE = 0x2A

3 bytes (0,0,0)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

32 bits (ip address, filled in on return)

32 bits (Netmask, filled in on return)

32 bits (Gateway, filled in on return)

32 bits (useDHCP, 0=no, 1=yes, filled in on return)

1.1.1.1.1.13 CMD_SET_IP_CONFIG

Set IP configuration – (SessionConfigIPAll struct– see Appendix A)

OPCODE = 0x2Bv

3 bytes (0,0,0)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

32 bits (ip address)
32 bits (Netmask)
32 bits (Gateway)
32 bits (useDHCP, 0=no, 1=yes)

1.1.1.1.1.14 CMD_QUERY_OPTIONS

Check system options – 88 bytes
OPCODE = 0x2C
3 bytes (0,0,0)
32 bytes (username, ASCII)
32 bytes (password, ASCII)
32 bits (HTTP enabled)
32 bits (Telnet enabled)
32 bits (Stream Options)
32 bits (Other Options)
32 bits (0)

DESCRIPTION:

32 bits (HTTP enabled, 0=no/off, TCP Port to use=yes/on, default port:80)
32 bits (Telnet enabled, 0=no/off, TCP Port to use=yes/on, default port:23)
32 bits (Stream Options)

1st byte = System Integrity (1=no/off, 0=yes/on) **Note: Inverse Logic**

2nd byte = System Kicker (0=no/off, else timing, see below)

bits in memory |_1_|_2_|_3_|_4_|_5_|_6_|_7_|_8_|

1st bit = (0 = minutes, 1 = hours)

2nd – 8th bits = how many minutes or hours.

3rd byte = SAP (0=no/off, 1=yes/on)

4th byte = BootStreaming (0=no/off, 1-100 seconds delay)

Note: BootStreaming query results must be divided by two (i.e. result = 140, the actual BootStreaming value is 70).

32 bits (Other Options)

1st byte = JoinOwnMcast (0=no/off, 1=yes/on)

2nd – 4th bytes ignored.

32 bits (0) Internal use.

1.1.1.1.1.15 CMD_SET_OPTIONS

Set system options – 88 bytes
OPCODE = 0x2D
3 bytes (0,0,0)
{option byte #2, 1-7, individual option items}
option 0: set all options
option 1: set web server only
option 2: set Telnet server only
option 3: set BootStreaming only
option 4: set System Integrity only
option 5: set System Kicker only
option 6: set SAP on/off only
option 7: set JoinOwnMcast only
32 bytes (username, ASCII)

32 bytes (password, ASCII)
 bytes in memory |_1_|_2_|_3_|_4_|

32 bits (HTTP enabled, 0=no/off, TCP Port to use=yes/on)
 all bytes (1-4) used for Web Server.

32 bits (Telnet enabled, 0=no/off, TCP Port to use=yes/on)
 all bytes (1-4) used for Telnet Server.

32 bits (Stream Options)
 1st byte = System Integrity (1=no/off, 0=yes/on) **Note: Inverse Logic**
 2nd byte = System Kicker (0=no/off, else timing, see below)
 bits in memory |_1_|_2_|_3_|_4_|_5_|_6_|_7_|_8_|
 1st bit = (0 = minutes, 1 = hours)
 2nd – 8th bits = how many minutes or hours.

3rd byte = SAP (0=no/off, 1=yes/on)
 4th byte = BootStreaming (0=no/off, 1-100 seconds delay)

32 bits (Other Options)
 1st byte = JoinOwnMcast (0=no/ff, 1=yes/on)
 2nd – 4th bytes ignored.

32 bits (0) Internal use.

1.1.1.1.1.16 CMD_QUERY_TTY

Check TTY settings - 220 bytes

OPCODE = 0x2F

3 bytes (0,0,0)

32 bytes (username, ASCII)

32 bytes (password, ASCII)

32 bits (TTY1 baudrate)

1 byte (TTY1 mode,

1 byte (TTY1 bits)

1 byte (TTY1 parity)

4 bits (TTY1 startbits)

4 bits(TTY1 stopbits)

32 bits (0), unused

16 bytes (0), unused

32 bits (TTY2 baudrate)

1 byte (TTY2 mode,

1 byte (TTY2 bits)

1 byte (TTY2 parity)

4 bits (TTY2 startbits)

4 bits(TTY2 stopbits)

32 bits (0), unused

16 bytes (0), unused

OPTIONS:

Byte #2=0x01, Launch AVN2XX RS-232 LoopBack Test, (Connector and port mode required)

Byte #2=0x02, Launch AVN2XX RS-422 LoopBack Test, (Connector and port mode required)

Byte #2=0x04, Launch AVN2XX GPIO LoopBack Test, (Connector required)

1.1.1.1.1.17 CMD_SET_TTY

Set TTY settings – 220 bytes

OPCODE = 0x30

3 bytes (0,0,0)

32 bytes (username, ASCII)
32 bytes (password, ASCII)
32 bits (TTY1 baudrate)
1 byte (TTY1 mode,
1 byte (TTY1 bits)
1 byte (TTY1 parity)
4 bits (TTY1 startbits)
4 bits(TTY1 stopbits)
32 bits (0), unused
16 bytes (0), unused
32 bits (TTY2 baudrate)
1 byte (TTY2 mode,
1 byte (TTY2 bits)
1 byte (TTY2 parity)
4 bits (TTY2 startbits)
4 bits(TTY2 stopbits)
32 bits (0), unused
16 bytes (0), unused
OPTIONS:
Byte #2=0x02, Set GPIO Parameters as specified in Byte #3

1.1.1.1.1.18 CMD_API_GET_VSTATS

Set OEM ID String – (API_VSTATS struct – see Appendix A)
OPCODE = 0x3A
3 bytes (0,0,0)
18 x 32bit Integers to be filled in with stats

1.1.1.1.1.19 CMD_SET_OEM_ID

Set OEM ID String – 196 bytes
OPCODE = 0x31
3 bytes (0,0,0)
32 bytes (username, ASCII)
32 bytes (password, ASCII)
64 bytes (string1, ASCII)
64 bytes (string2, ASCII)

1.1.1.1.1.20 CMD_FACTORY_DEFAULT

Reset factory defaults – (use SessionStopData struct – see Appendix A)
OPCODE = 0x32
3 bytes (0,0,0) = Manufacturer Factory Default
Same as CMD_VIDEO_STOP

OR

OPCODE = 0x32
3 bytes (1,0,0) = OEM Factory Default
Same as CMD_VIDEO_STOP

1.1.1.1.1.21 CMD_AUDIO_SEL_IN

OPCODE = 0x34
Next 3 Bytes(0x01,0,0)
audio[0] = audio input, 1=Balanced, 2=Unbalanced
username/password

1.1.1.1.1.22 CMD_AUDIO_SET_VOL

OPCODE = 0x35
To Set Volume:
Next 3 Bytes(0x01,0,0)
if(current audio source == Balanced)
 audio[3] = Balanced Volume Level
else
 audio[8] = Unbalanced Volume Level
username/password
To Set Pre-Amp:
Next 3 Bytes(0x0A,0,0)
if(current audio source == Balanced)
 audio[4] = Balanced Pre-Amp Level
else
 audio[9] = Unbalanced Pre-Amp Level
username/password
To Mute:
Next 3 Bytes(0x08,0,0)
username/password
To UnMute:
Next 3 Bytes(0x09,0,0)
username/password

1.1.1.1.1.23 CMD_AUDIO_SET_BAL

OPCODE = 0x36
Next 3 Bytes(0x01,0,0)
if(current audio source == Balanced)
 audio[2] = Balance Level for Balanced Input
else
 audio[7] = Balance Level for Unbalanced Input
username/password

1.1.1.1.1.24 CMD_AUDIO_SET_TONE

OPCODE = 0x37
Next 3 Bytes(0x01,0,0)
if(current audio source == Balanced)
 audio[1] = Tone Setting for Balanced Input
else

audio[6] = Tone Setting for Unbalanced Input
username/password

1.1.1.1.1.25 CMD_AUDIO_GET_ALL

OPCODE = 0x38
Next 3 Bytes(0x01,0,0)

Returns 12 Bytes of Audio Data plus the four command bytes:

cmd[1] = Mute (1=Muted, 0=Not Muted)
audio[0] = current input source (1=balanced, 2=unbalanced)
audio[1] = B Tone
audio[2] = B Balance
audio[3] = B Volume
audio[4] = B PreAmp
audio[5] = B Default
audio[6] = U Tone
audio[7] = U Balance
audio[8] = U Volume
audio[9] = U PreAmp
audio[10] = U Default
audio[11] = LED Code Mode

1.1.1.1.1.26 CMD_GIL_SET_LED_CODE

OPCODE = 0x39
To Set the LED Mode:
Next 3 bytes(0x01,0,0)
audio[11] contains LED Code value to be used
username/password

To Save Audio Settings:
Next 3 bytes(0x02,0,0)
username/password

To Default Audio Settings:
Next 3 bytes(0x03,0,0)
username/password

1.1.1.1.1.27 CMD_API_SAP_CONTROL

OPCODE = 0x3B
Next Byte = Specify Command (listing below)
0,0
username
password
...
command specific data in structure element

...

```
#define API_SAP_CONTROL_ENABLE      1
#define API_SAP_CONTROL_FREQ       2
#define API_SAP_CONTROL_SCOPE      5
#define API_SAP_CONTROL_NAME       6
#define API_SAP_CONTROL_INFO       7
#define API_SAP_CONTROL_KEYW       8
#define API_SAP_CONTROL_AUTH       9
#define API_SAP_CONTROL_COPY      10
#define API_SAP_CONTROL_EXTRA     11
#define API_SAP_CONTROL_SAVE      30
#define API_SAP_CONTROL_DEFAULT   31
#define API_SAP_CONTROL_DEFAULTS  32    // Set SAP Defaults and Save
```

1.1.1.1.1.28 CMD_ATOD_CONFIG

Configure Settings on the Analog to Digital Video Converter

OPCODE = 0x14

Options described below

(Next Byte & 0x01) Set Brightness

(Next Byte & 0x02) Set Contrast

(Next Byte & 0x04) Set Saturation

(Next Byte & 0x08) Set Hue

(Next Byte & 0x10) Set Closed Captions (requires stream restart)

(Next Byte & 0x20) Save AtoD Settings to Flash

(Next Byte & 0x40) Reset AtoD Settings to Default

Structure values set if above rules are met; see struct ATOD_API_STRUCT in Appendix A

1.1.1.1.1.29 CMD_STREAM_CONTROL

Configure Stream Destinations Filtering and Synchronization

OPCODE = 0x3C

Options described below

(Next Byte == 0x01) Get All Current Settings and Status

(Next Byte == 0x02) Start Main Stream

(Next Byte == 0x03) Start Aux Stream(s)

(Next Byte == 0x04) Stop Main Stream

(Next Byte == 0x05) Stop Aux Stream(s)

(Next Byte == 0x06) Set Main Stream Destination

(Next Byte == 0x07) Set Aux1 Stream Destination

(Next Byte == 0x08) Set Aux2 Stream Destination

(Next Byte == 0x09) Set Aux Stream Filter Mode

(Next Byte == 0x0A) Set Aux Stream Sync Mode

(Next Byte == 0x0B) Reset Aux Stream Defaults

(Next Byte == 0x0C) Save Aux Stream Settings

(Next Byte == 0x0D) Save Main Stream Settings

*NOTE main stream control should still be handled using CMD_START_VIDEO

Queries can be used with this command, just not sets.

Structure values set if above rules are met; see struct STREAM_API_STRUCT in Appendix A

GETs-

Start with an empty STREAM_API_STRUCT struct and fill in the "username" , "password" members. Next set the cmd[0] bit to 0x3C and send to device to get all the stream control data.

Getting the Filter Data out of the returned STREAM_API_STRUCT streamStruct;

1. The Streams statuses (Running or Stopped), including Auxiliary Stream(s) are determined by looking at the cmd[1] bit. If cmd[1] == 0 the Main/Primary Stream (PS) and both Auxiliary Streams are Stopped, else 1=PS Running, 3=PS and Aux1 running, 5=PS and Aux2 running, 7=All -PS, Aux1 and Aux2 running.

2. Take the data out of the STREAM_API_STRUCT (streamStruct) and do any network translation required.

```
int nFilterMode = ntohs(streamStruct.auxFmode);
```

3. The Audio Filter is either On or Off.

```
int nAudioFilterEnabled = nFilterMode & 0x10;
```

4. The Video Filter has three options (On/Normal, Skip a # of I Frames (Chunking), or Off/No Video). To get the Video Filter value two masks must be applied.

```
if( (nFilterMode & 0x100) > 0 ) // Video On/Normal
```

```
    // Do something with data.
```

```
else if( ( nFilterMode & 0x200 ) // Video Chunked
```

```
    // Do something with data.
```

```
else // Video Off
```

```
    // Do something with data.
```

5. Get the number of I Frames to skip by applying a mask and bit shifting. This number is set regardless of whether the I Frames will be skipped or not.

```
int nNumIFramesToSkip= (nFilterMode & 0xF000) >> 12;
```

1.1.1.1.1.30 CMD_API_PID_CONTROL

Configure Transport Stream PID Values

OPCODE = 0x15

Options described below

(Next Byte == 0x01) Query All PID Values

(Next Byte & 0x10) Update PMT PID

(Next Byte & 0x20) Update PCR PID

(Next Byte & 0x30) Update Video PID

(Next Byte & 0x40) Update Audio PID

Structure should be set accordingly

1.1.1.1.1.31 CMD_API_GOP_CONTROL

Configure Video Stream GOP Parameters

OPCODE = 0x16

Structure should be set with both values always

1.1.1.1.1.32 CMD_FIRMWARE_UPDATE

Upload new binary firmware file to AVN2XX unit

OPCODE = 0x01

3 stages described below

(1) Start Sequence: 6 bytes

OPCODE = 0x01

Next byte = 0x00 (Start)

Next 32 bits = Load Addr (0x00040020)

Start Responses:

Good 0x40

Bad C0 = Bad Memory location

(2) Data Sequence: 1460 bytes each

OPCODE = 0x01

Next byte = 0x02 (Data)

Next 16 bits = Src Pkt Number (start at 0)

Next 1456 bytes = binary data from file, final chunk can be smaller

Data Responses:

Good 0x42

Bad C2 = Bad Memory Location

Bad 00 = Packets out of order

(3) End Sequence: 6 bytes

OPCODE = 0x01

Next Byte = 0x03

Next 32 bits = CRC Checksum for file

End Responses:

Good 0x43

Bad 0x83

(4) Overwrite Firmware and Reset Unit: 6 bytes

OPCODE = 0x18

Next Byte = 0x00

Next 32 bits = (0x00000000)

Final Response:

Good 0x58

Bad 0x98

APPENDIX A (Control Structures)

```
typedef struct
{
    int    nIpAddr;           //32 bits
    int    nUdpPort;         //lower 16 bits
    int    bUseRTP;          //off=0, on=1
    int    bDoPacing;        //0
    int    bDoSync;         //0
    int    bUseKeepAlive;    //0
    int    nStreamBitrate;   //range 1200000-7500000 [15000000 AVN220]
    int    eVideoInput;      //Svideo=0, Comp=1
    int    eTvFormat;        //NTSC=0, PAL=1
    int    eVbr;             //0
    int    eVideoResolution; //D1=0,2/3D1=1,1/2D1=2,SIF=3
    int    eAudioBitRate;    //Off=0,256k=1,384k=2
    int    eStreamType;      //Transport Stream=2
    int    eErrorCorrection; //off=0, on=1 (only with RTP)
    int    nFecBurstSize;    //range 1-3
    int    nFecNumBursts;    //range 3-10
} SessionSettings;

typedef struct
{
    char    username[32];
    char    password[32];
} SessionAuthenticate;

typedef struct
{
    char    cmd[4];          // cmd[0] = CMD_QUERY_VIDEO
    SessionSettings settings;
} SessionQueryData;

typedef struct
{
    char    cmd[4];          // cmd[0] = CMD_START_VIDEO
    SessionAuthenticate authenticateData;
    SessionSettings settings;
} SessionStartData;

typedef struct
{
    char    cmd[4];          // cmd[0] = CMD_STOP_VIDEO
    SessionAuthenticate authenticateData;
} SessionStopData;

typedef struct
{
    char    cmd[4];          // cmd[0] = CMD_VERSION_VIDEO
    char    version[80];    // firmware version info
} SessionVersionData;
```

```
typedef struct
{
    char          cmd[4];          // cmd[0] = CMD_RESET
    SessionAuthenticate authenticateData;
} SessionResetData; /* Also used for StopLock */
```

```
typedef struct
{
    char          cmd[4];          // cmd[0] = CMD_QUERY_LOCK
    char          lock_status[4]; // lock_status[0] : 0=unlocked, 1=locked
} SessionQueryLockData;
```

```
typedef struct
{
    char          cmd[4];
    char          Port;
    int           BaudRate;
    SessionAuthenticate authenticateData;
} SessionCamTypeData;
```

```
typedef struct
{
    char          cmd[4];
    char          din[3];
    char          dout;
    SessionAuthenticate authenticateData;
} SessionDigIOData;
```

```
typedef struct
{
    char          cmd[4];
    SessionAuthenticate authenticateData;
    int           nIpAddr;
    int           nNetmask;
    int           nGateway;
    int           nDHCP;          // 0=off, 1=on
} SessionConfigIPAll;
```

```
typedef struct
{
    char          cmd[4];
    unsigned char audio[12];
    SessionAuthenticate authenticateData;
} SessionAudioData;
```

```
typedef struct
{
    char          cmd[4];
    char          username[32];
    char          password[32];
    int           enabled;
    int           frequency;
    int           owner;
    int           id;
    int           scope;
    char          session_name[32];
}
```

```

    char    session_info[32];
    char    keywords[32];
    char    author[32];
    char    copyright[32];
    char    extra[128];
}SAP_API_STRUCT;

```

```

typedef struct
{
    char cmd[4];
    char username[32];
    char password[32];
    unsigned char brightness;
    unsigned char contrast;
    unsigned char saturation;
    unsigned char hue;
    unsigned char closedCap;
    unsigned char reserved[7];
}ATOD_API_STRUCT;

```

```

typedef struct
{
    char          cmd[4];
    char          username[32];
    char          password[32];
    unsigned int  mainAddr;
    unsigned int  auxAddr1;
    unsigned int  auxAddr2;
    unsigned short mainPort;
    unsigned short auxPort1;
    unsigned short auxPort2;
    unsigned int  auxFmode;    //Filter Mode
    unsigned char auxSmode;    //Sync Mode
    unsigned char action;
    unsigned char reserved[8];
}STREAM_API_STRUCT;    //100 bytes

```

```

typedef struct
{
    char          cmd[4];
    SessionAuthenticate SA;
    unsigned short pmt_pid;
    unsigned short pcr_pid;
    unsigned short vid_pid;
    unsigned short aud_pid;
}API_PID_DATA;

```

```

typedef struct
{
    char          cmd[4];    // cmd[0] = CMD_API_GOP_CONTROL
    SessionAuthenticate SA;
    unsigned char gop_distance;
    unsigned char gop_length;
} API_GOP_DATA;

```

```

typedef struct _VIDEO_INPUT_STATS
{
    int      vbuf_overwrites;      //0 # of buffer overwrites
    int      last_7114_status;     //1 status register mask (8)
    int      INTL_7114;           //2 interlace detection
    int      HLVLN_7114;          //3 horiz and vert loops locked
    int      GLIMT_7114;          //4 luminance gain MAX
    int      GLIMB_7114;          //5 luminance gain MIN
    int      WIPA_7114;           //6 White Peak loop active
    int      NORDY_7114;          //7 Video Not Ready
    int      last_6752_status;     //8 status register mask (16)
    int      NOVID_6752;          //9 no input video signal
    int      VSYNC_6752;          //10 diff in conf and detected
    int      OBUFF_6752;          //11 output buffer overflow
    int      I2CXX_6752;          //12 illegal I2C-bus Command
    int      GNERR_6752;          //13 General Error
    int      AUDIO_6752;           //14 audio problems
    int      MMSYN_6752;          //15 memory manager re-sync
    int      restart_count;        //16 # of SIT() restarts
    int      ts_sync_lost;         //17 # of TS sync losses
}VIDEO_INPUT_STATS;

typedef struct _API_VSTATS
{
    char cmd[4];
    VIDEO_INPUT_STATS vis;
} API_VSTATS;

```

APPENDIX B (OPCODES/DATA)

```
#define ACK_BIT          0x40 // SUCCESS Indicator
#define ERR_BIT         0x80 // ERROR Indicator

#define CMD_FIRMWARE_UPDATE 0x01
#define CMD_RESET          0x07 //(SessionResetData)
#define CMD_QUERY_VIDEO    0x11 //(SessionQueryData)
#define CMD_START_VIDEO    0x12 //(SessionStartData)
#define CMD_STOP_VIDEO     0x13 //(SessionStopData)
#define CMD_CONFIG_ATOD    0x14 //(ATOD_API_STRUCT)
#define CMD_API_PID_CONTROL 0x15 //(API_PID_DATA)
#define CMD_API_GOP_CONTROL 0x16 //(API_GOP_DATA)
#define CMD_VERSION_VIDEO  0x23 //(SessionVersionData)
#define CMD_STOP_LOCK      0x24 //(SessionResetData)
#define CMD_QUERY_LOCK     0x25 //(SessionQueryLockData)
#define CMD_QUERY_AUTH     0x26 //(SessionResetData)
#define CMD_SET_CAM_TYPE   0x27 //(SessionCamTypeData)
#define CMD_QUERY_DIG_IN   0x28 //(SessionDigIOData)
#define CMD_SET_DIG_OUT    0x29 //(SessionDigIOData)
#define CMD_GET_IP_CONFIG  0x2A //(SessionConfigIPAll)
#define CMD_SET_IP_CONFIG  0x2B //(SessionConfigIPAll)
#define CMD_QUERY_OPTIONS  0x2C
#define CMD_SET_OPTIONS    0x2D
#define CMD_QUERY_TTY      0x2F
#define CMD_SET_TTY        0x30
#define CMD_SET_OEM_ID     0x31
#define CMD_FACTORY_DEFAULT 0x32 //(SessionStopData)
#define CMD_AUDIO_SEL_IN   0x34 //(SessionAudioData)
#define CMD_AUDIO_SET_VOL  0x35 //(SessionAudioData)
#define CMD_AUDIO_SET_BAL  0x36 //(SessionAudioData)
#define CMD_AUDIO_SET_TONE 0x37 //(SessionAudioData)
#define CMD_AUDIO_GET_ALL  0x38 //(SessionAudioData)
#define CMD_GIL_SET_LED_CODE 0x39 //(SessionAudioData)
#define CMD_API_SAP_CONTROL 0x3B //(SAP_API_STRUCT)
#define CMD_API_STREAM_CONTROL 0x3C //(STREAM_API_STRUCT)
```


APPENDIX D (Camera Control)

The AVN2XX is equipped with two serial interfaces (RS232 and RS422) which can be set to operate in several different functional modes. If the AVN2XX is configured for Camera Control then it will create a uni-directional connection between a serial port and a TCP port listening for connections. This is very similar to the Passive Tunneling functionality, except data only flows in one direction, and a different TCP port is used.

For RS232 based camera control, address TCP port 6104

For RS422 based camera control, address TCP port 6103

APPENDIX E (SA Camera Control)

The AVN2XX is equipped with two serial interfaces (RS232 and RS422) which can be set to operate in several different functional modes. If the AVN2XX is configured for SA Camera Control then it will create a uni-directional connection between a serial port and a TCP port listening for connections. This is very similar to the Camera Control functionality mode, except rather than transparently passing data; it expects predefined data commands as input.

Using SA Camera Control mode it is possible to control a variety of cameras using the AVN2XX with no knowledge of the underlying control protocol. Currently supported control protocols include PelcoD, PelcoP, and Visca.

For RS232 based SA camera control, address TCP port 6110

For RS422 based SA camera control, address TCP port 6109

Data Structure (Binary not ASCII)

```
{
  unsigned 8bit Camera_Control_Type; //1=PelcoD, 2=PelcoP, 3=Visca
  unsigned 8bit Camera Address;      //unique camera address (1-255)
  unsigned 8bit Camera Command;     //listing below
  unsigned 8bit Specify Speeds;     //0=no (ignore), 1=yes (use them)
  unsigned 32bit Serial Data Ptr;    //used internally (ignore)
  unsigned 32bit Pan Speed;          //horizontal movement
  unsigned 32bit Tilt Speed;         //vertical movement
}
```

Command Listing:

0	UNDEFINED
1	PAN RIGHT
2	PAN LEFT
3	TILT UP
4	TILT DOWN
5	STOP MOVEMENT
6	ZOOM IN
7	ZOOM OUT
8	STOP ZOOM

Once the AVN2XX has been configured for a SA CamControl mode and restarted it will listen on the port specified above for the particular Serial Port. It is now possible for an external application to connect to that socket via TCP and send commands specified in the above format to trigger camera control activity.