Data Structures:

Abbreviations used:

$:	signifies hexadecimal number
ADDR:	device address
CHNUM:	channel number code
COMSTAT:	communications status code
DCL:	device control language
DT:	Device-type code (predefined)
ID:	Manufacturer's ID code (predefined)
MEMNUM:	memory number code
OPSTAT:	operational status code
RPE 228:	Dual channel programmable one-third octave equalizer
SPL:	stored parameter list (product dependent)

RPE 228 Stored Parameter List (SPL):

Index	**Name**	**Encoding Method** (see next section)
0	31.5 Hz band	Gain 1
1	40 Hz band	Gain 1
2	50 Hz band	Gain 1
3	63 Hz band	Gain 1
4	80 Hz band	Gain 1
5	100 Hz band	Gain 1
6	125 Hz band	Gain 1
7	160 Hz band	Gain 1
8	200 Hz band	Gain 1
9	250 Hz band	Gain 1
10	315 Hz band	Gain 1
11	400 Hz band	Gain 1
12	500 Hz band	Gain 1
13	630 Hz band	Gain 1
14	800 Hz band	Gain 1
15	1 kHz band	Gain 1
16	1.25 kHz band	Gain 1
17	1.6 kHz band	Gain 1
18	2 kHz band	Gain 1
19	2.5 kHz band	Gain 1
20	3.15 kHz band	Gain 1
21	4 kHz band	Gain 1
22	5 kHz band	Gain 1
23	6.3 kHz band	Gain 1
24	8 kHz band	Gain 1
25	10 kHz band	Gain 1
26	12.5 kHz band	Gain 1
27	16 kHz band	Gain 1
28	input gain	Gain 2
29	output gain	Gain 2
30	mute	OFF = 0, ON = 1 (muted)
31	low cut	Low Cut
32	high cut	High Cut
33	bypass	OFF = 0, ON = 1 (bypassed)

Encoding Methods for RPE 228 Stored Parameter List:

Gain 1 (0.5 dB steps):
$00 = 0.0 dB, $01 = +0.5 dB, $02 = +1.0 dB, etc., $14 (maximum value) = +10.0 dB
$FF = –0.5 dB, $FE = –1 dB, etc., $EC (minimum value) = –10 dB
(Two's complement for negative gain values)

Gain 2 (1 dB steps):
$00 = 0 dB, $01 = +1 dB, $02 = +2 dB, etc., $0C (maximum value) = +12 dB
$FF = –1 dB, $FE = –2 dB, etc., $F4 (minimum value) = –12 dB
(Two's complement for negative gain values)

Low Cut (10 Hz steps):
$00 (minimum value) = OFF, $01 = 10 Hz, $02 = 20 Hz, etc., $14 (maximum value) = 200 Hz

High Cut (1 kHz steps):
$00 (minimum value) = OFF, $01 = 20 kHz, $02 = 19 kHz, etc., $14 (maximum value) = 1 kHz

RPE 228 Global Parameters:

Type:	Bytes:	Description:
Unit name	16	NULL terminated ASCII string if less than 16 characters long. Otherwise, omit NULL.
Ch 1 name	16	NULL terminated ASCII string if less than 16 characters long. Otherwise, omit NULL.
Ch 2 name	16	NULL terminated ASCII string if less than 16 characters long. Otherwise, omit NULL.
Unit lock flag	1	1 if unit is locked (read-only)
Elapsed time	4	Time of use in seconds (read-only) (Note: This is unsigned long integer. If bit 31 is set, it means that an error occurred, e.g. someone removed the EEPROM while the unit was powered, and that time was restarted from that point.
Reserved	4	Normally set to 0,0,0,0 (factory use only)

Device address (ADDR):

Valid address range is 1 through 250 (0, 251, 252, 253, 254, and 255 are reserved)

Device-type code (DT):

$00 = dual channel one-third octave equalizer (for Rane RPE 228)

Manufacturer's identification code (ID):

$08 = Rane Corporation

Channel number codes (CHNUM):

$01 = device channel #1, $02 = device channel #2, etc. (RPE 228 has two channels)

Memory number codes (MEMNUM):

$00 = *live* or working memory, $01 = preset memory #1, $02 = preset memory #2, ... , $10 = preset memory #16 (RPE 228 has 16 preset memories per channel)

Communications status codes (COMSTAT):

$00 = no error
$01 = invalid data
$02 = invalid command code
$03 = device locked
$04 = device *not* locked
$05 = channel(s) muted

$06 = channel(s) *not* muted
$07 = checksum error

Operational status codes (OPSTAT):
00 = no error

RW 232 Commands:

Send data to channel (81 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $28 (number of data bytes to follow including checksum)
Transmit $81 (command code)
Transmit CHNUM
Transmit MEMNUM
Transmit 2 bytes; starting param byte index (See Note 5)
Transmit SPL
Transmit Checksum
Get COMSTAT

Program channel from memory (82 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $04 (number of data bytes to follow including checksum)
Transmit $82 (command code)
Transmit CHNUM (See Note 4)
Transmit MEMNUM
Transmit Checksum
Get COMSTAT

Program all channels of all devices from memory (82 hex):
Transmit $FB $00 $FB $00
Transmit $00 $03 (number of data bytes to follow including checksum)
Transmit $82 (command code)
Transmit MEMNUM
Transmit Checksum

Lock device (85 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $85 (command code)
Transmit $79 (Checksum)
Get COMSTAT

Unlock device (86 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)

Transmit $86
Transmit $78 (Checksum)
Get COMSTAT

Mute channel (87 hex):

Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $03 (number of data bytes to follow including checksum)
Transmit $87 (command code)
Transmit CHNUM (See Note 4)
Transmit Checksum
Get COMSTAT

Mute all channels of all devices (87 hex):

Transmit $FB $00 $FB $00
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $87 (command code)
Transmit $77 (Checksum)

Unmute channel (88 hex):

Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $03 (number of data bytes to follow including checksum)
Transmit $88 (command code)
Transmit CHNUM (See Note 4)
Transmit Checksum
Get COMSTAT

Unmute all channels of all devices (88 hex):

Transmit $FB $00 $FB $00
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $88 (command code)
Transmit $76 (Checksum)

Get OPSTAT (00 hex):

Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $00 (command code)
Transmit $FE (Checksum)
Get OPSTAT
Get memory source number for channel 1
Get memory source number for channel 2
Get working/stored flag (See Note 6)
Get working/dirty flag (See Note 7)
Get Checksum
Get COMSTAT

Flash COM LEDs on all units (00 hex):
Transmit $FB $00 $FB $00
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $00 (command code)
Transmit $FE (Checksum)

Get data from channel (01 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $08 (number of data bytes to follow through start param)
Transmit $01(command code)
Transmit CHNUM
Transmit MEMNUM
Transmit 2 bytes; starting param byte (See Note 5)
Transmit 2 bytes; number of param bytes (See Note 5)
Transmit Checksum (See Note 3)
Get SPL
Get Checksum (for SPL)
Get COMSTAT

Get device-type and manufacturer's identification codes (02 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $02 (command code)
Transmit $FC (Checksum)
Get COMSTAT

Send globals (8C hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $3B (number of data bytes to follow including checksum)
Transmit $8C (command code)
Transmit Global Parameters
Transmit Checksum
Get COMSTAT

Get globals (03 hex):
Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $03 (command code)
Transmit $FB $FB (Checksum, See Note 1)
Get Global Parameters
Get Checksum (for Global Parameters)
Get COMSTAT

Get device serial/identification number (04 hex):

Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $04 (command code)
Transmit $FA (Checksum)
Get 3-byte number, MSB first
Get Checksum (for serial ID)
Get COMSTAT

Get software revision (05 hex):

Transmit ADDR header ($FB xx FB xx, where xx = ADDR)
Get DT
Get ID
Transmit $00 $02 (number of data bytes to follow including checksum)
Transmit $05 (command code)
Transmit $F9 (Checksum)
Get hardware revision
Get firmware revision (× 10)
Get Checksum (for hardware and firmware revisions)
Get COMSTAT

Notes:

1. When the value $FB occurs anywhere except in an ADDR header, it is repeated.
2. The data size is the number of bytes, prior to the $FB repetition, between the command code and the checksum inclusively.
3. The checksum applies to the "data size" bytes through the byte immediately before the checksum, inclusive. Repeated $FB's are counted only once. The sum is the twos complement negative of the LS Byte of the arithmetic sum. For example, if the sum is $1234, the checksum is $CC.
4. CHNUM is normally 1 or 2 for a 2-channel device. A CHNUM of 0 is allowed for this message and means both channels.
5. The parameter bytes are indexed using a 2-byte number (MSB first) starting with 0. The number of parameter bytes also uses a 2-byte number with the same format. When sending parameters, the number sent is determined by the data size.
6. The working/stored flag is set if the working memory for either channel doesn't match the stored memory from which it originated.
7. The working/dirty flag is set when the RPE is powered up, or when a memory is recalled. It is cleared when the working parameters are sent or received.

RW 232 Communications Interface:

RW 232 is loosely based on PA-422. One key hardware difference is that RW 232 does not utilize hardware handshaking via DTR/DSR. The beginning of a message always takes the form:

$FB xx FB xx (where xx = ADDR)

Note: *When $FB appears in the body of the message, it is always repeated.*

Input port:	9-pin female input port (DB-9F) on device
Output port:	9-pin male output port (DB-9M) on device (for serial linking to the input port on the next device. Up to 250 devices can be linked in this manner.)
Device address means:	8−position DIP switch on device (valid device addresses are 1 through 250)
Baud Rate:	19.2 kilobaud
Character frame bits:	1 start bit, 8 data bits, 1 parity bit (even), and 1 stop bit
Cabling:	Use standard RS-232 serial printer or modem cables.
	Warning: *NULL modem cables will not work!*
Host or computer interface:	Standard PC serial COM port (DB-9M, or DB-25M with adapter)
	Note: *Only three lines, Tx, Rx, and Ground, are used.*

References:

Rodgers, Robert L., "PA-422 Communications Interface and Device Control Language", Journal of the Audio Engineering Society, Vol. 38, Number 9, 1990 September, pp. 619−639.

Audio Engineering Society, Inc., "AES Recommended practice for sound-reinforcement systems−Communications interface (PA-422)", Journal of the Audio Engineering Society, Vol. 39, Number 9, 1991 September, pp. 664−679.

Standards documents: AES15-1991 (Audio Engineering Society)
ANSI S4.49-1991 (American National Standards Institute)

```
//---------------------------------------------------------------------------
//              Example Packet Expansion code for RW 232 Messages
//
//              Rane Corporation
//
//  09-10-96   -   Devin Cook (Derived from RW232.CPP code)
//---------------------------------------------------------------------------

// This code only deals with the Body of an RW 232 message (Command/Data)
//
// The steps needed to fully communicate an RW 232 are:
//
// 1. Send the Address:  [FB xx FB xx]
// 2. Get the returned Device Type and Device ID flags
// 3. Send the FB expanded Body
// 4. Get and check the returned ComStat byte

// Take a simple command and expand it into a full packet.
//
// Input:
//              Buff            - BYTE array with the unexpanded message and lots of extra room
//              MsgLen          - Unexpanded message length
//
// Steps required are:
// 1. Add Packet size.  This is simply the Command length + 1 for the checksum
// 2. Duplicate 0xFBs
// 3. Calculate Checksum
// 4. Add Checksum to packet (Check for a 0xFB Checksum!)
// 5. Copy Packet back to the buffer
// 6. Return the new Packet Size
//
// Note: The buffer must be large enough to accept the expanded data.
//      No checking is done to verify it is, so be careful!

// A packet into this routine consists of the one byte Command and any Data

int CmdToPacket(BYTE Buff[], int MsgLen)
{
        BYTE L_MSB = ((MsgLen+1) >> 8) & 0xFF;          // Grab MSB of Size
        BYTE L_LSB =  (MsgLen+1) & 0xFF;                // Grab LSB of Size

// FBs is the number of 0xFB bytes in the messages
        int FBs = 0;

// Don't forget to check message length for FBs
        if (L_MSB == 0xFB)
```

```
                FBs ++;

        if (L_LSB == 0xFB)
                FBs ++;

// Calculate Checksum of Message Length bytes along with bytes in the packet
        int CheckSum = L_MSB + L_LSB;

        for (int x=0;x<L;x++)
        {
                CheckSum += Buff[x];
                if (Buff[x] == 0xFB)
                        FBs ++;
        }

        CheckSum = (256-CheckSum) & 0xFF;

// Don't forget to up the FB count for a FB checksum!
        if (CheckSum == 0xFB)
                FBs ++;

// New Length is:
// 2 (For 2 length bytes) +
// L (Old message Length) +
// Repeated FBs count     +
// 1 (For CheckSum)

        int NewLen = 2 + MsgLen + FBs + 1;

// Create a temporary holding tank for Packet Expansion
        BYTE Packet[MAX_CMD_BUF];

        BYTE *Ptr = Packet;

// Stick message length in the packet (Watching for FBs of course)
        *(Ptr++) = L_MSB ;
        if (L_MSB == 0xFB)
                *(Ptr++) = 0xFB;

        *(Ptr++) = L_LSB ;
        if (L_LSB == 0xFB)
                *(Ptr++) = 0xFB;

// Expand the original packet into the new buffer
        for (x=0;x<L;x++)
        {
```

```
                *(Ptr++) = Buff[x];
                if (Buff[x] == 0xFB)
                *(Ptr++) = 0xFB;
        }

// Add the Checksum byte (or Bytes if Checksum == FB)
        *(Ptr++) = CheckSum ;
        if (CheckSum == 0xFB)
                *(Ptr++) = 0xFB;

// Copy the expanded packet back into the original buffer
        memcpy(Buff,Packet,NewLen);

        return NewLen;
}
```