Controlling ZonePRO™ products with RS-232

This document is outlined as follows:

Introduction

The purpose of this document is to show how to connect to a ZonePRO product, generate and format command strings, calculate a checksum, and receive feedback using the dbx ZonePRO protocol. Refer to the "Full Duplex Data Link Specification.pdf" for full specifications. This quick guide is for developers who want to rapidly achieve communications with a ZonePRO using an open loop approach. While a closed loop driver would be desired, some developers may choose to implement an open loop driver. This document will assist developing an open loop driver to control most aspects of a ZonePRO.

Section 1 – Getting Started/Basic Command Structure

Baud Rate

The ZonePRO is fixed at**: 57.6** kbps, 8N1.

Big Endian

Multi Byte data types are sent Big Endian, which means they are send most significant byte first. If a 16 bit word 0x1234 is sent, it is presented to the transmit register as 0x12 first then 0x34.

Data Types

UBYTE	8 bits unsigned	0 - 255
UWORD	16 bits unsigned	0 - 65535
ULONG	32 bits unsigned	0 - 4294967295

Resync Request / Resync Acknowledge

The communications protocol uses special characters to synchronize both ends of the serial connection and to keep the connection open. To synchronize the serial drivers, Resysnc Request and Resync Acknowledge are used.

Resynch_Request	0xFF
Resysnc_Acknowledge	0xF0

Since these characters are not accepted from the rest of the protocol, the resynch procedure will issue a string of Resync_Request and Resync_Acknowledge bytes to flush the receiving state machine. To synchronize with a ZonePRO send 16 Resynch_Request and 261 Resysnc_Acknowledge bytes.

Ping
To maintain a connection, a PING byte lets the other side know that you are still connected.

Ping	0xF0 0x8C

If a message has not been sent within the last second, send a PING. A timeout of 2.5 seconds will result in the ZonePRO attempting a resync.

Resync_Acknowledge Byte
Send this byte at the beginning of **every command**. This byte will keep the ZonePRO from attempting a resync when using open-loop implementation.

Sync	0xF0

Frame Start Bytes
To indicate the start of a frame, two bytes are used. The frame bytes will always precede a Message Header.

Frame_Start	0x64
Frame_Count	0x00, or 0x01-0xFF

For an open loop implementation, use a Frame_Count of 0x00. This indicates to the receiver that it need not acknowledge the receipt of the message.

Message Header
The Message Header is 21 bytes indicating the source and destination of the message, the version of the frame format, and the message type. All messages have the following header:

Field	Size	Comments
Protocol Version	BYTE	Currently set to 1, this will be used mainly to indicate a difference in the header or
Length	ULONG	Size of message from Version to the end of the data payload, in bytes.
Source	[UWORD:ULONG]	[Virtual Device : Object] address of the source. Object address of 0 indicates the Virtual Device.
Destination	[UWORD:ULONG]	[Virtual Device : Object] address of the destination. Object address of 0 indicates the Virtual Device.
Message ID	UWORD	Disco message = 0x0000 Set message = 0x0100 Recall Scene = 0x9001 SubscribeAll = 0x0113 UnsubcribeAll = 0x0114 Get message = 0x0103 Get Object List = 0x011E
Flags	UWORD	Bit 0 = ReqAck Bit 1 = Ack Bit 2 = Information Bit 3 = Error Bit 4 = Event Bit 8 – 15 = Hop Count

Object ID

The object ID is a 4 byte address of a particular processing or virtual device (Router, Mixer, etc). This value will need to known in order to control any State Variable (SV) inside the device in question, such as selecting a Router input or controlling a Mixer fader. The easiest way to get the Object ID in *decimal* format is to click on the device in the ZonePRO program screen and press <Ctrl>+<Shift>+<o>. Note that bit 0 is presented at the top and bit 3 at the bottom of the dialog box. When sending this Object ID in a command, send b3,b2,b1,b0. Also, see Section 3 – Using the Network Trace Window for another easy way to get Object values in hex.

Basic Command Structure (Unacknowleged – Open Loop)

Frame Start	*UBYTE*	*0x64*
Frame Count	*UBYTE*	*0x00*
VERSION	UBYTE	0x01
LEN	ULONG	Length of entire packet (not including FS FC, CS)
SRC	UWORD:ULONG	[Device : Object]

DEST	UWORD:ULONG	`[Device : Object]`
MSG_ID	UWORD	Specific type of command issued
FLAGS	UWORD	(there is no guaranteed bit)
(Payload)		
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

Payload
The payload size and content is specific to the type of message being sent and contains the information to complete a task specified by the Message ID. Some messages have no payload at all. More information will be given on Payload contents for each specific command. In general, the Payload will take on a structure similar to this:

Number SVs	UWORD	How many SVs are being changed
SV_ID	UWORD	Which control within the Object
Data type	UBYTE	What datatype you are sending in SV_Val
SV_Val	(varies)	Value to set the SV

Number SVs
How many SVs are being changed within one Frame.

SV_ID
Each control inside an object is assigned a particular SV (state variable) ID. For example, the fader inside a mixer, a particular source inside a router, or a mute button.

Data_Type
The data type held by the SV_Val. See Appendix for table of all possible Data Types.

Data_Type = 1 for a single unsigned byte of range 0 to 255 (ie. a Mute button, or Route)
Data_Type = 3 for two unsigned bytes of range 0 to 65535 (ie. a fader)

SV_Val
This variable holds the value of the SV_ID in question. For example, if a Router is the object we want to control, the SV_Val will be a single byte of a value 0(OFF), 1 (Lobby Mic), 2 (Phone Page), etc. The SV_Val for a fader within a Mixer object consists of two bytes and will hold a value between 0-19F (hex) [0-415 (decimal)].

CCIT checksum
The CCIT checksum is an 8 bit CRC byte used to validate the Header and Payload. This byte can be calculated by initializing the checksum byte to 0xFF and passing each byte

though the following calculation. The Network_CCITT_8_Table is included in the Appendix.

```
UCHAR update_bcc(UCHAR current_bcc, UCHAR new_byte)
{
        return Network_CCITT_8_Table[current_bcc ^ new_byte];
}
```

The checksum is calculated on all bytes in the Frame, Header, and Payload. See Section 4 for detailed information concerning Checksum calculations.

Section 2 – Setting Up and Maintaining a Communication Connection

In order to send commands to a dbx ZonePRO unit from a 3rd party controller, a connection should be kept alive by the **continuous** sending and receiving of specific commands. There are two different levels of "keep-alive" or "heartbeat" commands that should be **continuously** exchanged. The first is a Ping designed to keep the transmission layer connection open. This is done by sending a 0xF0 0x8C every 1 second. The ZonePRO will respond with a 0x8C. *If the user desires to only send commands and not receive any feedback from the ZonePRO, then only this connection must be maintained* (HACK: If the 3rd party controller is not capable of parsing commands, storing independent variables, or receiving commands, then the user may send pre-formed commands with a 0xF0 preceding all commands. This will open a connection just long enough for the ZonePRO to accept a command).

The second is a Disco command and it is designed to keep a connection open at the Protocol layer. *If a user wants to receive feedback from the ZonePRO*, a Disco must be sent from the 3rd party controller at least every 10 seconds to keep this connection open. To initiate a connection with the ZonePRO, a Disco Broadcast should be sent. Any ZonePRO devices on this connection will answer back.

The Disco Broadcast command:

Frame Start	*UBYTE*	*0x64*
Frame Count	*UBYTE*	*0x00*
VERSION	UBYTE	0x01
LEN	ULONG	0x00000017
SRC	UWORD:ULONG	0xNODE00000000
DEST	UWORD:ULONG	0xFFFF00000000
MSG_ID	UWORD	0x0000 (Disco)

FLAGS	UWORD	0x0004 (Info)
(Payload) SRC Node	 UWORD	 0x0033
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

When using the serial port use the source node address 0x00 0x33. So the actual command sent out the comm port is:

0xF0 0x64 0x00 0x01 0x00 0x00 0x00 0x17 0x00 0x33 0x00 0x00 0x00 0x00 0xFF 0xFF 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x33 0x20(checksum byte)

The ZonePRO will respond to this Disco message with its address (node). Parse this incoming message in order to get the ZonePRO's address. Once the address is found you must reply to all incoming Disco messages (send at least every 10 seconds whether a Disco is received or not) or the connection will be shut down. The reply is the same message as the broadcast command, but with the ZonePRO's address inserted in place of the 0xFF 0xFF. The incoming Disco message will be:

VERSION	UBYTE	0x01
LEN	ULONG	Length of entire packet
SRC	UWORD/ULONG	0xNODE00000000
DEST	UWORD/ULONG	0xFFFF00000000
MSG_ID	UWORD	0x0000 DiscoInfo
FLAGS	UWORD	0x0004 (Info)
PAYLOAD…		
Node	UWORD	Node address of sender
Cost	UBYTE	Aggregated cost of route back to source
Serial Number	BLOCK	Sender's Serial Number
Max Message Size	ULONG	Max Msg size sender can handle
NetworkID	UBYTE	Network type of sender 0x01 = TCP/IP 0x02 = RS485 0x03 = USB 0x04 = RS232

NetworkInfo	Network specific	Network specific info of sender
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

It will be assumed that the dbx node address is 0x00 0x20 (decimal 32). So here is a sample of an incoming Disco command:

0x64 0x00 0x010x00 0x00 0x00 0x2F 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x33 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x20 0x05 0x00 0x10 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0F 0xD7 0x00 0x48 0xB0 0x10 0x00 0x00 0x00 0x04 0x(checksum)

So the 3rd party controller needs to send the following command for every incoming Disco (send at least every 10 seconds whether a Disco is received or not):

0xF0 0x64 0x00 0x01 0x00 0x00 0x00 0x17 0x00 0x33 0x00 0x00 0x00 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x04 0x00 0x33 0x66(checksum byte)

Once these two "heartbeats" are being continuously exchanged, the connection is open and the 3rd party controller can send and receive commands as necessary.

Guaranteed Acknowledgement

Every command that is sent or received starts with the 0xFS 0xFC start frame bytes. If the Frame Count (second byte) is anything besides a 0x00, then it is a guaranteed service message and a 0xA5 must be sent to acknowledge this message has been received. The ZonePRO will try and initiate a re-sync if it doesn't receive the 0xA5 ack for every guaranteed message sent (HACK: the user may send a 0xA5 after every received message if the user doesn't want to deal with guaranteed service requirements).

Resync

The ZonePRO will send at least 261 0xFF's if it believes it is out of sync with the controller. The ZonePRO will not accept any serial commands in this state. This happens when the 0xF0 0x8C heartbeat, or the Disco commands, are not being sent at required intervals. This will also happen if a 0xA5 is not received for each guaranteed packet sent.

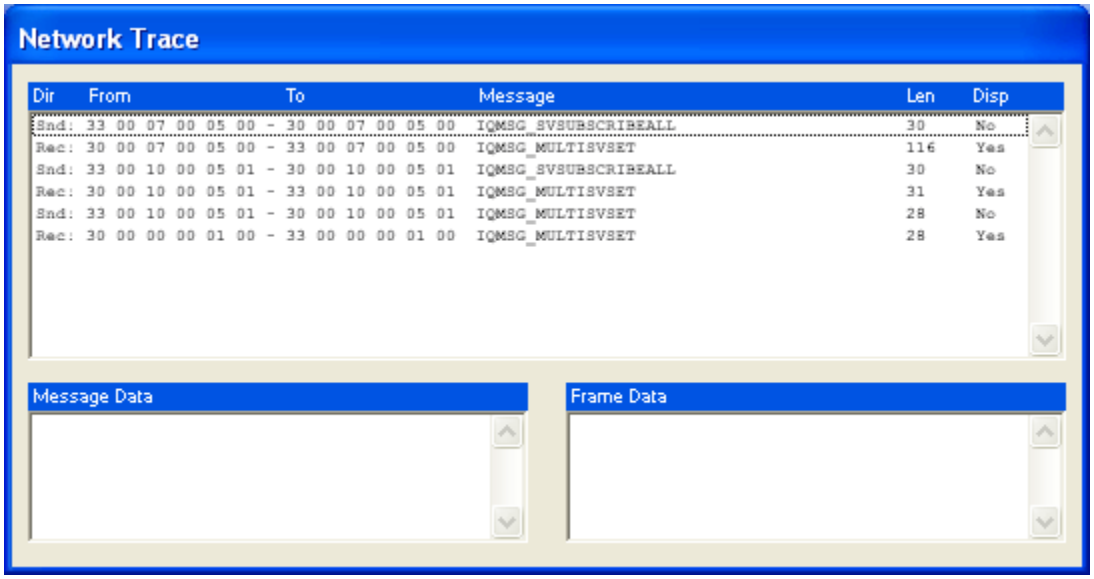
Section 3 – Generating Command Strings via Network Trace Window

Depending on the configuration currently in use inside the ZonePRO, the Object ID for each virtual device (ie. Router, Mixer, etc.) will be different from the example strings. Using the Network Trace Window is an easy way to get the exact strings for your particular ZonePRO configuration. Nearly every control can be replicated by a 3rd party controller using this method.

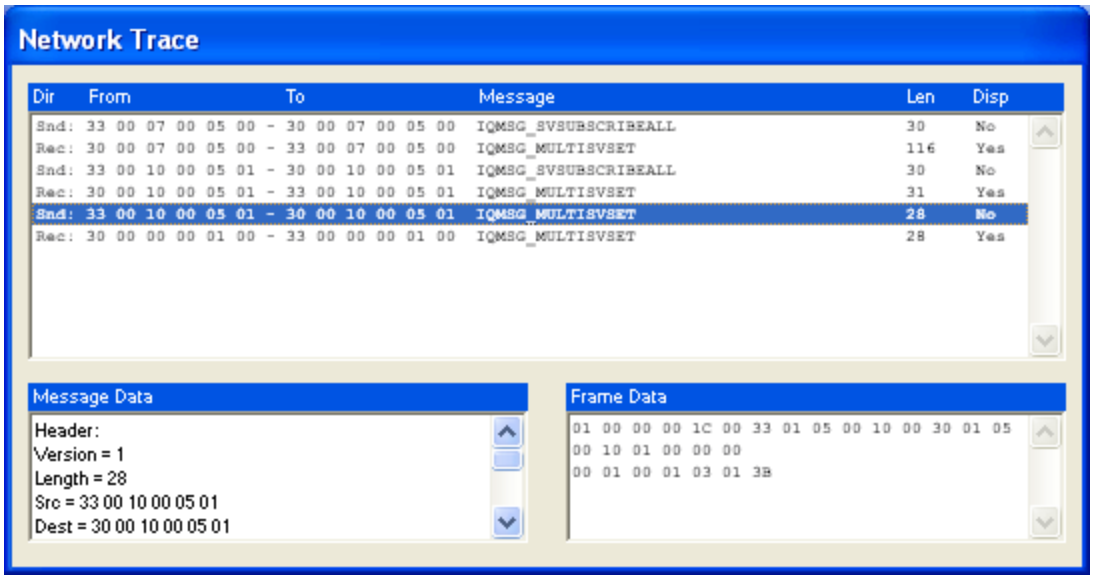
Trace Window

With the ZonePRO Designer connected with a ZonePRO, bring the "ZonePRO Designer" window into focus.

<Ctrl><Shift><T> will bring up a Network Trace window. All communication between the ZonePRO Designer and ZonePRO will be detailed in this window.



If you click on an individual message, the data from this message will be printed in the Message Data and Frame Data windows.



Highlighting the bytes in the Frame Data window and copying it to your clipboard can be a useful way of "stealing" strings from the ZonePRO Designer to put into a controller code (*please note that the message prefaced with "Snd:" is highlighted*).

In order to use the following hexadecimal string (Object ID in bold on following line):

01 00 00 00 1C 00 33 **01 05 00 10** 00 30 **01 05 00 10** 01 00 00 00 00 01 00 01 03 01 3B

You need to add a Frame Start, Frame Count and Checksum byte.

64 00 01 00 00 00 1C 00 33 01 05 00 10 00 30 01 05 00 10 01 00 00 00 00 01 00 01 03 01 3B **96**

Using this method, any string that ZonePRO designer sends to the ZonePRO can be duplicated by a custom controller.

Add an extra Resync_Acknowledge

If a communication error has occurred, the ZonePRO may be trying to resync. Since a controller in open loop is not looking for these Resync Requests, an extra Resync_Acknowledge at the start of every Frame will get the ZonePRO receiver out of resync mode.

F0 64 00 01 00 00 00 1C 00 33 01 05 00 10 00 30 01 05 00 10 01 00 00 00 00 01 00 01 03 01 3B 96

This is also advised for the PING byte that is sent every second. **0xF0 0x8C**

IMPORTANT NOTE: If you are trying to control a Mixer object you will need to overcome an error in the Network Trace Window. The error is that the SV_ID bytes are shown swapped in the window (ie. If the SV_ID bytes are displayed 0D 00 then you need to form the string as 00 0D). If you are controlling a fader you need to worry about the

same network trace error in the SV_ID and the SV_Val bytes. The range of SV_Val for a fader is 0-19F (hex) [0-415 (decimal)]. Here is a string setting the Chan 1 Mixer Lobby Fader to the maximum value:

F0	64	0	1	0	0	0	1C	0	33	1	5	0	1E	0		
20	1	5	0	1E	1	0	5	0	0	1	0	0	3	1	9F	B5

Useful note:
Using the mouse to move a slider results in many messages making it difficult to capture the strings that you want to use. With the cursor, highlight the state variable that you wish to create a string for and enter in the value desired. In this case, "10" was entered into the Master Gain state variable and this value was sent to the ZonePRO.

MultiSVSet message – 0x0100 (Message ID)
Most changes that a controller will make on a ZonePRO involve a MultiSVSet message. State Variables (SV) control gains, input selection or other settings that a controller may change. The following example will set 3 different SV values within the same Object. The format of the command is:

The MultiSVSet command:

Frame Start	*UBYTE*	*0x64*
Frame Count	*UBYTE*	*0x00*
VERSION	UBYTE	0x01
LEN	ULONG	0x000000xx <xx is variable>
SRC	UWORD:ULONG	0xNODEVDOBJECT
DEST	UWORD:ULONG	0xNODEVDOBJECT
MSG_ID	UWORD	0x0100
FLAGS	UWORD	0x0000
(Payload) Number SVs SV_ID Data type SV_Val SV_ID Data type SV_Val SV_ID Data type SV_Val	 UWORD UBYTE (varies) UWORD UBYTE (varies) UWORD UBYTE (varies)	 Number of SVs being changed in this frame The state variable in the object What datatype you are sending Value to set the SV The state variable in the object What datatype you are sending Value to set the SV The state variable in the object What datatype you are sending Value to set the SV
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

In order to send a MultiSVSet message to a state variable you need to know what object it is contained in – specified by the destination in the frame, which SV_ID the variable is identified by in the object, and the value that you are writing to the state variable. If more than one SV is to be changed, SV_ID, Data type and Message ID can be repeated. The following is an example of using the MultiSVSet to control the input sources on a **Router**: (all values in HEX) [used on default configuration of ZP1260]

F0 64 00 [SB FB FC] (Send in front of ALL commands)
01 [Version]
00 00 00 xx [Length of packet]
00 33 [Use this address for a serial 3rd Party Controller]

01 05 xx xx [Obj ID: varies with configuration loaded in ZonePRO] (get using Network Trace)
xx xx [ZonePRO address] (ZP 1260 default \x00\x20)
01 05 xx xx [Obj ID: varies with configuration loaded in ZonePRO] (use same bytes as previous)
01 00 [Message ID] (this is used for all SET commands)
05 00 [Flag]
00 01 [Num SV's being changed in this frame]
00 00 [SV_ID]
01 [Data Type]
xx [SV_Val] (Source Input: 0=none, 1=Lobby Mic, 2= PhonePage, 3 = CD_L, etc)
xx [CheckSum: See Section 4]

Sample String: Change Ch.3 Router's input to Phone Page
F0 64 00
01
00 00 00 1B
00 33
01 05 02 20 (For channel 3. If it was for channel 2 then: 01 05 01 1F; If for chan1: 01 05 00 1E)
00 20
01 05 02 20 (same bytes as above)
01 00
05 00
00 01
00 00
01
02 [This selects input. If no source was desired then 00. Lobby Mic = 01]
26

Recall Scene: 0x9001 (Message ID)

To change a scene on a ZonePRO, issue a recall scene command. If you are using subscriptions, the changed parameter values will come back to you in multisvset messages.

VERSION	UBYTE	0x01
LEN	ULONG	0x00000017
SRC	UWORD:ULONG	0xNODEVDOBJECT
DEST	UWORD:ULONG	0xNODEVDOBJECT
MSG_ID	UWORD	0x9001,
Flags	UWORD	0x0000
(PAYLOAD) **Scene**	[UWORD]	Scene to load, 0 (default), 1 , 2, 3…
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

Section 4 – Calculating Checksums

This document will show two ways of calculating checksums. The first method is using code to perform the checksum operations systematically. The second method is using a Microsoft Excel Spreadsheet and manually entering commands into the spreadsheet. The spreadsheet will then calculate the checksum.

How to calculate a checksum using code for the dBX ZonePRO:

```
//shown in AMX Netlinx syntax
//CCIT copied from "Full Duplex Data Link Specification.pdf"
//$ = hex
CHAR CCIT[] = {$5E,$BC,$E2,$61,$3F,$DD,$83,$C2,$9C,$7E,$20,$A3,$FD,$1F,$41,
        $9D,$C3,$21,$7F,$FC,$A2,$40,$1E,$5F,$01,$E3,$BD,$3E,$60,$82,$DC,
        $23,$7D,$9F,$C1,$42,$1C,$FE,$A0,$E1,$BF,$5D,$03,$80,$DE,$3C,$62,
        $BE,$E0,$02,$5C,$DF,$81,$63,$3D,$7C,$22,$C0,$9E,$1D,$43,$A1,$FF,
        $46,$18,$FA,$A4,$27,$79,$9B,$C5,$84,$DA,$38,$66,$E5,$BB,$59,$07,
        $DB,$85,$67,$39,$BA,$E4,$06,$58,$19,$47,$A5,$FB,$78,$26,$C4,$9A,
        $65,$3B,$D9,$87,$04,$5A,$B8,$E6,$A7,$F9,$1B,$45,$C6,$98,$7A,$24,
        $F8,$A6,$44,$1A,$99,$C7,$25,$7B,$3A,$64,$86,$D8,$5B,$05,$E7,$B9,
        $8C,$D2,$30,$6E,$ED,$B3,$51,$0F,$4E,$10,$F2,$AC,$2F,$71,$93,$CD,
        $11,$4F,$AD,$F3,$70,$2E,$CC,$92,$D3,$8D,$6F,$31,$B2,$EC,$0E,$50,
        $AF,$F1,$13,$4D,$CE,$90,$72,$2C,$6D,$33,$D1,$8F,$0C,$52,$B0,$EE,
        $32,$6C,$8E,$D0,$53,$0D,$EF,$B1,$F0,$AE,$4C,$12,$91,$CF,$2D,$73,
        $CA,$94,$76,$28,$AB,$F5,$17,$49,$08,$56,$B4,$EA,$69,$37,$D5,$8B,
        $57,$09,$EB,$B5,$36,$68,$8A,$D4,$95,$CB,$29,$77,$F4,$AA,$48,$16,
        $E9,$B7,$55,$0B,$88,$D6,$34,$6A,$2B,$75,$97,$C9,$4A,$14,$F6,$A8,
        $74,$2A,$C8,$96,$15,$4B,$A9,$F7,$B6,$E8,$0A,$54,$D7,$89,$6B,$35};

//COMMAND COPIED FROM NETWORK TRACE WINDOW
CHAR DBX[] = {$64,$00,$1,$0,$0,$0,$1B,$00,$33,$1,$5,$0,$1E,$0,$20,$1,$5,$0,$1E,
        $1,$0,$5,$0,$0,$1,$0,$0,$1,$1};


BUTTON_EVENT[dvTP,65]
{
PUSH:
   {
   LOCAL_VAR CHAR BCC; //CHECKSUM
   LOCAL_VAR INTEGER I; //LOOP COUNTER

   BCC = $FF;  //INITIALIZE CHECKSUM

   FOR (I = 1; I<= LENGTH_ARRAY(DBX);I++)
        {
        BCC = CCIT[(BCC BXOR DBX[I])]; //bitwise XOR each element of command
        }
   }

}
```

Checksum (BCC) is $08 for the DBX[] command stated in the example;

Microsoft Excel Spreadsheet Method

Download the "ZonePROChecksumCalc.xls" spreadsheet from http://www.dbxpro.com/Download/index.htm If all the checksum boxes are full of #####, then go to Tools, Add-Ins and check the Analysis ToolPak and Analysis Toolpak-VBA options. This spreadsheet has many example strings taken from the default configuration. The default configuration has all output channel devices as Routers. At the end of each

command is the checksum denoted by an outlined box. If your values do not match the example values, enter your string into the document and the checksum will be calculated.

Copy the string taken from the network trace window and paste it into the spreadsheet (remember to change the source address to 0x00 0x33 if using the comm port) and the correct checksum will be calculated.

Section 5 – Feedback

The ZonePRO utilizes a command called 'subscribe' to manage all feedback traffic. Once a subscribe command has been issued, the ZonePRO will send a string containing the current state of the SV (or SV's) every time the SV changes value. This will allow a 3rd party controller to get SV updates when a user changes values via the front panel, ZonePRO GUI, or from another 3rd party controller. The ZonePRO will continue to send feedback until a 'unsubscribe' command is issued, or until the unit looses power.

SVSubscribeAll

This message is used to subscribe to every State Variable under an Object or Virtual Device. If this message is sent to a Mixer Object, then a feedback string will be generated if a user changes any values within that particular Mixer Object.

The SVSubscribeAll command:

Frame Start	*UBYTE*	*0x64*
Frame Count	*UBYTE*	*0x00*
VERSION	UBYTE	0x01
LEN	ULONG	0x000000xx <xx is variable>
SRC	UWORD:ULONG	0xNODEVDOBJECT
DEST	UWORD:ULONG	0xNODEVDOBJECT
MSG_ID	UWORD	0x0113
FLAGS	UWORD	0x0000
(Payload) Subscriber Address Subscription Type Sensor Rate	 UWORD:ULONG UBYTE UWORD	 0xNODEVDOBJECT 0-ALL, 1 –Non-Sensor, 2- Sensor Period in milliseconds
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

Sample String: Subscribing to output channel 1 Router on a ZP640 (source address 0x0033, dest address 0x0030):

0x64 0x00 0x01 0x00 0x00 0x00 0x20 0x00 0x33 0x01 0x05 0x00 0x10 0x00 0x30 0x 01 0x 05 0x00 0x10 0x01 0x13 0x00 0x00 0x00 0x33 0x01 0x05 0x00 0x10 0x01 0x00 0x00 0x00 0x01 0x16

SVUnSubscribeAll

This message is used to unsubscribe to every State Variable under an Object or Virtual Device.

The SVUnSubscribeAll command:

Frame Start	*UBYTE*	*0x64*
Frame Count	*UBYTE*	*0x00*
VERSION	UBYTE	0x01
LEN	ULONG	0x000000xx <xx is variable>
SRC	UWORD:ULONG	0xNODEVDOBJECT
DEST	UWORD:ULONG	0xNODEVDOBJECT
MSG_ID	UWORD	0x0114
FLAGS	UWORD	0x0000
(Payload) Subscriber Address Subscription Type	 UWORD:ULONG UBYTE	 0xNODEVDOBJECT 0-ALL, 1 –Non-Sensor, 2- Sensor
Checksum	*UBYTE*	CCITT-8 (over FS, FC, Header, Payload)

Note: If the ZonePRO issues a resync request, that means the connection has been shut down, and all SV's will need to be re-subscribed. All SV's will need to be subscribed to whenever the AC power is lost to the ZonePRO unit.

Appendix

IP Connections

All information contained above for RS232 connections is valid for TCP/IP connections except that no FS, FC, or checksum characters are transmitted within a command. Also, the TCP/IP connection also does not need any PING, ACK/NAK, RESYNC/RESYNC_ACK, or Guaranteed ACK because UDP and TCP have their own link layer. The IP port is 3804.

SV ID's for a mixer object. All mixers will have these same SV ID's, the object ID is the same for all SV's contained within the mixer. Each mixer device will have its own object ID.

Lobby Mic Gain = \x00\x00
PhonePage Gain = \x00\x01
CD L Gain = \x00\x02
CD R Gain = \x00\x03
Satellite L Gain = \x00\x04
Satellite R Gain = \x00\x05
Jukebox L Gain = \x00\x06
Jukebox R Gain = \x00\x07
TV L Gain = \x00\x08
TV R Gain = \x00\x09
DVD L Gain = \x00\x0A
DVD R Gain = \x00\x0B
Master Fader Gain = \x00\x0C
Master Mute = \x00\x0D

SV ID's for a router object.

Input Source SV = \x00\x00 for all router input sources.
Master Fader SV = \x00\x01
Master Mute SV = \x00\x02

SV ID's for unit input gains.

All unit input gains have an SV of \x00\x00. The object ID will change for each input.

Example Strings:

These strings were collected from the default configuration of ZP640. The Object ID in the first string is in bold.

```
/* UCHAR Ch1LobyMic[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x1, 0x27},
```

```
/* UCHAR Ch1phonePg[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x2, 0xC5},

/* UCHAR Ch1CD[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x3, 0x9B},

/* UCHAR Ch1Satelite[]*/
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x4, 0x18},

/* UCHAR Ch1TV[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x5, 0x46},

/* UCHAR Ch1Jukebox[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x6, 0xA4},

/* UCHAR Ch2LobyMic[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x1, 0xD7},

/* UCHAR Ch2phonePg[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x2, 0x35},

/* UCHAR Ch2CD[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x3, 0x6B},

/* UCHAR Ch2Satelite[]*/
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x4, 0xE8},

/* UCHAR Ch2TV[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x5, 0xB6},

/* UCHAR Ch2Jukebox[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x6, 0x54},

/* UCHAR Ch3LobyMic[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x1, 0xDE},

/* UCHAR Ch3phonePg[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x2, 0x3C},

/* UCHAR Ch3CD[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x3, 0x62},

/* UCHAR Ch3Satelite[]*/
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x4, 0xE1},

/* UCHAR Ch3TV[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x5, 0xBF},
```

```
/* UCHAR Ch3Jukebox[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x6, 0x5D},

/* UCHAR Ch4LobyMic[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x1, 0x2E},

/* UCHAR Ch4phonePg[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x2, 0xCC},

/* UCHAR Ch3CD[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x3, 0x92},

/* UCHAR Ch3Satelite[]*/
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x4, 0x11},

/* UCHAR Ch3TV[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x5, 0x4F},

/* UCHAR Ch3Jukebox[] */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1B, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0x1, 0x6, 0xAD},

/* UCHAR m20dB[]       */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x0F, 0x8D},

/* UCHAR m18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x23, 0xD },

/* UCHAR m16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x37, 0xF1},

/* UCHAR m14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x4B, 0xAA},

/* UCHAR m12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x5F, 0x56},

/* UCHAR m10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x73, 0xD6},

/* UCHAR m08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x87, 0xC3},

/* UCHAR m06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x9B, 0xFD},

/* UCHAR m04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xAF, 0x22},
```

```
/* UCHAR m02dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xC3, 0xE4},

/* UCHAR z00dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xD7, 0x18},

/* UCHAR p02db[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xEB, 0x5 },

/* UCHAR p04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xFF, 0xF9},

/* UCHAR p06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x13, 0x77},

/* UCHAR p08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x27, 0xA8},

/* UCHAR p10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x3B, 0x96},

/* UCHAR p12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x4F, 0xF },

/* UCHAR p14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x63, 0x8F},

/* UCHAR p16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x77, 0x73},

/* UCHAR p18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x8B, 0xA4},

/* UCHAR p20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x0, 0x10, 0x0, 0x30,
0x1, 0x5, 0x0, 0x10, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x9F, 0x58},

/* UCHAR m20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x0F, 0xF9},

/* UCHAR m18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x23, 0x79},

/* UCHAR m16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x37, 0x85},

/* UCHAR m14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x4B, 0xDE},
```

```
/* UCHAR m12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x5F, 0x22},

/* UCHAR m10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x73, 0xA2},

/* UCHAR m08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x87, 0xB7},

/* UCHAR m06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x9B, 0x89},

/* UCHAR m04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xAF, 0x56},

/* UCHAR m02dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xC3, 0x90},

/* UCHAR z00dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xD7, 0x6C},

/* UCHAR p02db[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xEB, 0x71},

/* UCHAR p04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xFF, 0x8D},

/* UCHAR p06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x13, 0x3 },

/* UCHAR p08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x27, 0xDC},

/* UCHAR p10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x3B, 0xE2},

/* UCHAR p12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x4F, 0x7B},

/* UCHAR p14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x63, 0xFB},

/* UCHAR p16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x77, 0x7 },

/* UCHAR p18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x8B, 0xD0},
```

```
/* UCHAR p20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x1, 0x11, 0x0, 0x30,
0x1, 0x5, 0x1, 0x11, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x9F, 0x2C},

/* UCHAR m20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x0F, 0x65},

/* UCHAR m18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x23, 0xE5},

/* UCHAR m16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x37, 0x19},

/* UCHAR m14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x4B, 0x42},

/* UCHAR m12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x5F, 0xBE},

/* UCHAR m10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x73, 0x3E},

/* UCHAR m08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x87, 0x2B},

/* UCHAR m06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x9B, 0x15},

/* UCHAR m04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xAF, 0xCA},

/* UCHAR m02dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xC3, 0xC },

/* UCHAR z00dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xD7, 0xF0},

/* UCHAR p02db[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xEB, 0xED},

/* UCHAR p04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xFF, 0x11},

/* UCHAR p06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x13, 0x9F},

/* UCHAR p08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x27, 0x40},
```

```
/* UCHAR p10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x3B, 0x7E},

/* UCHAR p12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x4F, 0xE7},

/* UCHAR p14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x63, 0x67},

/* UCHAR p16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x77, 0x9B},

/* UCHAR p18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x8B, 0x4C},

/* UCHAR p20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x2, 0x12, 0x0, 0x30,
0x1, 0x5, 0x2, 0x12, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x9F, 0xB0},

/* UCHAR m20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x0F, 0x11},

/* UCHAR m18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x23, 0x91},

/* UCHAR m16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x37, 0x6D},

/* UCHAR m14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x4B, 0x36},

/* UCHAR m12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x5F, 0xCA},

/* UCHAR m10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x73, 0x4A},

/* UCHAR m08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x87, 0x5F},

/* UCHAR m06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0x9B, 0x61},

/* UCHAR m04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xAF, 0xBE},

/* UCHAR m02dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xC3, 0x78},
```

```
/* UCHAR z00dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xD7, 0x84},

/* UCHAR p02db[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xEB, 0x99},

/* UCHAR p04dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x0, 0xFF, 0x65},

/* UCHAR p06dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x13, 0xEB},

/* UCHAR p08dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x27, 0x34},

/* UCHAR p10dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x3B, 0xA },

/* UCHAR p12dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x4F, 0x93},

/* UCHAR p14dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x63, 0x13},

/* UCHAR p16dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x77, 0xEF},

/* UCHAR p18dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x8B, 0x38},

/* UCHAR p20dB[]      */
{0xF0, 0x64, 0x0, 0x1, 0x0, 0x0, 0x0, 0x1C, 0x0, 0x33, 0x1, 0x5, 0x3, 0x13, 0x0, 0x30,
0x1, 0x5, 0x3, 0x13, 0x1, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x1, 0x3, 0x1, 0x9F, 0xC4},
};
```

CRC 8 bit calculation – 8 bit

Bcc(new char) = Network_CCITT_8_Table[(UCHAR)(bcc ^ (new_char))]

Network_CCITT_8_Table[] =

```
0x00,0x5E,0xBC,0xE2,0x61,0x3F,0xDD,0x83,0xC2,0x9C,0x7E,0x20,0xA3,0xFD,0x1F,0x41,

0x9D,0xC3,0x21,0x7F,0xFC,0xA2,0x40,0x1E,0x5F,0x01,0xE3,0xBD,0x3E,0x60,0x82,0xDC,

0x23,0x7D,0x9F,0xC1,0x42,0x1C,0xFE,0xA0,0xE1,0xBF,0x5D,0x03,0x80,0xDE,0x3C,0x62,
```

```
0xBE,0xE0,0x02,0x5C,0xDF,0x81,0x63,0x3D,0x7C,0x22,0xC0,0x9E,0x1D,0x43,0xA1,0xFF,
0x46,0x18,0xFA,0xA4,0x27,0x79,0x9B,0xC5,0x84,0xDA,0x38,0x66,0xE5,0xBB,0x59,0x07,
0xDB,0x85,0x67,0x39,0xBA,0xE4,0x06,0x58,0x19,0x47,0xA5,0xFB,0x78,0x26,0xC4,0x9A,
0x65,0x3B,0xD9,0x87,0x04,0x5A,0xB8,0xE6,0xA7,0xF9,0x1B,0x45,0xC6,0x98,0x7A,0x24,
0xF8,0xA6,0x44,0x1A,0x99,0xC7,0x25,0x7B,0x3A,0x64,0x86,0xD8,0x5B,0x05,0xE7,0xB9,
0x8C,0xD2,0x30,0x6E,0xED,0xB3,0x51,0x0F,0x4E,0x10,0xF2,0xAC,0x2F,0x71,0x93,0xCD,
0x11,0x4F,0xAD,0xF3,0x70,0x2E,0xCC,0x92,0xD3,0x8D,0x6F,0x31,0xB2,0xEC,0x0E,0x50,
0xAF,0xF1,0x13,0x4D,0xCE,0x90,0x72,0x2C,0x6D,0x33,0xD1,0x8F,0x0C,0x52,0xB0,0xEE,
0x32,0x6C,0x8E,0xD0,0x53,0x0D,0xEF,0xB1,0xF0,0xAE,0x4C,0x12,0x91,0xCF,0x2D,0x73,
0xCA,0x94,0x76,0x28,0xAB,0xF5,0x17,0x49,0x08,0x56,0xB4,0xEA,0x69,0x37,0xD5,0x8B,
0x57,0x09,0xEB,0xB5,0x36,0x68,0x8A,0xD4,0x95,0xCB,0x29,0x77,0xF4,0xAA,0x48,0x16,
0xE9,0xB7,0x55,0x0B,0x88,0xD6,0x34,0x6A,0x2B,0x75,0x97,0xC9,0x4A,0x14,0xF6,0xA8,
0x74,0x2A,0xC8,0x96,0x15,0x4B,0xA9,0xF7,0xB6,0xE8,0x0A,0x54,0xD7,0x89,0x6B,0x35,
```

Data Type

The data types used in the MultiSVSet message are found in the table below.

Name	'C' Declaration	Range	Bytes	Value
BYTE	char	-128 to 127	1	0
UBYTE	unsigned char	0 to 255	1	1
WORD	Short	-32768 to 32767	2	2
UWORD	unsigned short	0 to 65535	2	3
LONG	long	-2147,483648 to 2147,483647	4	4
ULONG	unsigned long	0 to 4,294,967,926	4	5
FLOAT32	float	As IEEE-754	4	6
FLOAT64	double	As IEEE-754	8	7
BLOCK	N/A	0 to 65535 bytes	N/A	8
UNISTRING	N/A	0 to 65535 bytes	N/A	9
LONG64	N/A	Very Big ;-)	8	10
ULONG64	N/A	Huge :-)	8	11