RS-232 Control
of the
Advantage PMX84

Biamp Systems, 14130 N.W. Science Park, Portland, Oregon 97229 U.S.A. (503) 641-7287
an affiliate of Rauland-Borg Corp.

Introduction

This document contains technical information relating to computer control of the Biamp Advantage PMX84 Programmable Matrix Switch. This information is intended for advanced users - in particular for those who wish to develop their own computer programs to control the PMX84. It is assumed that the reader is an experienced programmer and has some familiarity with standard programming practices, binary and hexadecimal numbers, the ASCII character set, asynchronous serial data communications, and RS-232 interfaces.

Hexadecimal, ASCII-Hex, and "Pseudo-Hex" Numbers

Throughout this document, hexadecimal numbers shall be represented by preceding the number with "0x". For example: the hexadecimal equivalent of the decimal number 255 is 0xFF.

Individual ASCII characters, except control characters, will be enclosed in single quotes. For example: the ASCII character 'A' has the hexadecimal value 0x41. The ASCII "carriage return" control character shall be represented as ↵. An ASCII code chart is included with this document for your convenience.

When an 8-bit binary data value is being transmitted over a serial data communications link, it is a common practice to transmit the byte as two "ASCII-hex" characters - one character represents the most significant nibble of the data value and the other character represents the least significant nibble (a nibble is 4-bits; half of a byte). Each ASCII-hex character is in the range of '0' thru '9' or 'A' thru 'F' (from the ASCII code chart, 0x30 thru 0x39 or 0x41 thru 0x46). For example, the *hexadecimal* equivalent of the decimal number 61 is 0x3D. To transmit this in an *ASCII-hex* format, first transmit the ASCII character '3' (whose hex value is 0x33), followed by the ASCII character 'D' (whose hex value is 0x44). This is the standard way to transmit an ASCII-hex value. In some cases, the lower case characters 'a' thru 'f' (0x61 thru 0x66) are accepted in addition to the upper case letters 'A' thru 'F'.

The Advantage PMX84 does not utilize standard ASCII-hex format. The PMX84 computer commands implement what I call "pseudo-hex" notation (also sometimes referred to as a "poor man's" ASCII-hex notation). Instead of representing the hexadecimal value using the ASCII characters '0' thru '9' and 'A', 'B', 'C', 'D', 'E', and 'F', the pseudo-hex format uses the ASCII characters '0' thru '9' and the characters '**:**', '**;**', '<', '=', '>', and '**?**'. As you can see by studying the ASCII code chart, a binary nibble may be converted to its equivalent pseudo-hex character by simply adding 0x30.

A binary/decimal/hexadecimal/pseudo-hex conversion chart is included at the end of this manual for your convenience.

Data Communications Parameters

Serial Port Data Communications Parameters

The PMX84 normally communicates through its standard RS-232 serial interface at a data rate ("baud" rate) of 2400 bits per second with 8 data bits, no parity, and 1 stop bit. Although the factory default setting is 2400 bits per second, operation at 9600 bits per second is also possible by changing an internal jumper strap. The PMX84 utilizes a subset of the standard 7-bit ASCII character set. The eighth data bit (bit 7 - the most significant bit) of each character transmitted by the computer should always be 0. The computer should not echo the characters it receives from the PMX84.

Since the PMX84's standard RS-232 serial interface only has a single-character input buffer for its incoming serial data, a form of flow control must be implemented by the computer in order to guarantee that no characters are lost. Neither hardware (DTR or RTS) nor XON/XOFF (also known as DC1/DC3 or control-S/control-Q) handshaking is supported by the PMX84. However, each character which the PMX84 receives with its standard RS-232 serial interface will be "echoed" back to the computer. Flow control may be implemented by the computer software by simply waiting for each character's echo from the PMX84 before transmitting the next character, since the PMX84 doesn't retrieve and echo an incoming character until it has finished processing the previous character.

PMX-Link Data Communications Parameters

The PMX-Link is a serial communications interface which was designed primarily to allow multiple PMX84s to be linked together in such a way that all units react to the same remote control button events. Although the PMX-Link is not a true RS-232 interface, the voltage levels and impedances associated with the PMX-Link are compatible with RS-232. The PMX-Link operates at a bit rate of 2400 bits per second with 8 data bits, no parity, and 1 stop bit. The PMX-Link bit rate is not adjustable.

The PMX-Link is a one-way communications link. The PMX84 does not echo the characters it receives and there is no flow control mechanism. The PMX-Link has a small (8 byte) input buffer which allows it to receive short bursts of data at the full data rate (approximately one character every 4.2 msec), however, when performing computer control using the PMX-Link, the computer should avoid sending characters to the PMX84 at an *average* rate of greater than approximately 20 characters per second (approximately one character every 50 msec).

Computer Control

The Advantage PMX84 has an RS-232-compatible serial interface which allows it to be controlled by a computer or by a system controller such as those provided by AMX or Crestron. In addition to its standard RS-232 serial interface, the PMX84 has an RS-232 compatible expansion port ("PMX-Link") which may also be used for computer control. The PMX84 offers the following three methods of computer control:

- Control Button Emulation. This method of computer control allows the computer to emulate Biamp's standard infrared remote control transmitter or wall-mount remote control panel. Using this method, the computer outputs ASCII characters which are equivalent to the characters which would be generated by a remote control connected to remote port 1 of the PMX84. These ASCII characters are transmitted from the computer to the PMX84's standard RS-232 compatible serial port. Control Button Emulation is simple to perform, however, it only provides "one-way" control of the PMX84 - it allows the computer to send simple commands *to* the PMX84, but it does not provide any mechanism for requesting status information *from* the PMX84.

- PMX-Link. This method of computer control is similar in concept to Control Button Emulation, however, in addition to emulating a remote control connected to remote port 1, the PMX-Link allows emulation of all four remote ports plus all sixteen logic inputs of the PMX84. Using this method, the computer or controller transmits binary "serial key codes" to the PMX84 using the RS-232 compatible PMX-Link. Computer control using the PMX-Link is simple to perform, however, like Control Button Emulation, it only provides "one-way" control of the PMX84 - it allows the computer to send simple commands *to* the PMX84, but it does not provide any mechanism for requesting status information *from* the PMX84. As a general rule, the PMX-Link should be used for computer control only in installations where the four remote control ports and sixteen logic inputs are *not* being used.

- Advanced Computer Control. This method of computer control provides advanced commands which allow "two-way" control of the PMX84. Using Advanced Computer Control commands, the computer may request status information *from* the device as well as send commands *to* the device. The computer communicates with the PMX84 using the PMX84's standard RS-232 compatible serial port.

Control Button Emulation

Control Button Emulation is the simplest form of computer control of the Advantage PMX84. This method of operation allows the computer to emulate the operation of a standard Biamp remote control transmitter connected to remote port 1.

For each button on a standard Biamp remote control, there is a corresponding ASCII character. In order to emulate a remote control button, the computer simply transmits the corresponding ASCII character to the PMX84's standard RS-232 serial port. Each character received by the PMX84 will be echoed back to the computer.

The standard Biamp remote control devices never exceed a transmission rate of 9 characters per second. If the computer wishes to perform Control Button Emulation at a rate of greater than 20 characters per second (50 msec per character), flow control should be implemented by waiting for the echo of each character before transmitting the next character. At slower speeds, flow control should not be necessary.

The following table summarizes the ASCII character codes for Control Button Emulation corresponding to each of the 40 remote control buttons for remote 1 of the PMX84. These button codes are also summarized on the ASCII code chart provided at the end of this manual. The remote control buttons on the standard Biamp transmitter are numbered from left to right going from bottom to top with the lower left-hand button being button number 1.

Control Button Emulation ASCII Codes (remote 1)

button 1	'B'	(0x42)	button 21	'V'	(0x56)
button 2	'C'	(0x43)	button 22	'W'	(0x57)
button 3	'D'	(0x44)	button 23	'X'	(0x58)
button 4	'E'	(0x45)	button 24	'Y'	(0x59)
button 5	'F'	(0x46)	button 25	'Z'	(0x5A)
button 6	'G'	(0x47)	button 26	'['	(0x5B)
button 7	'H'	(0x48)	button 27	'\'	(0x5C)
button 8	'I'	(0x49)	button 28	']'	(0x5D)
button 9	'J'	(0x4A)	button 29	'^'	(0x5E)
button 10	'K'	(0x4B)	button 30	'_'	(0x5F)
button 11	'L'	(0x4C)	button 31	'`'	(0x60)
button 12	'M'	(0x4D)	button 32	'b'	(0x62)
button 13	'N'	(0x4E)	button 33	'c'	(0x63)
button 14	'O'	(0x4F)	button 34	'd'	(0x64)
button 15	'P'	(0x50)	button 35	'e'	(0x65)
button 16	'Q'	(0x51)	button 36	'f'	(0x66)
button 17	'R'	(0x52)	button 37	'g'	(0x67)
button 18	'S'	(0x53)	button 38	'h'	(0x68)
button 19	'T'	(0x54)	button 39	'i'	(0x69)
button 20	'U'	(0x55)	button 40	'j'	(0x6A)

Device Select Prefix Characters

When using Advanced Computer Control, up to eight PMX84s may be linked together and individually controlled by the computer (if each device is first assigned a unique device number). When using Control Button Emulation, a limited subset of device addressing may be performed, which allows individual control of up to four PMX84s (with device numbers 1 thru 4). This is accomplished by transmitting a device select prefix code immediately prior to each control button ASCII character code. The device select prefix code is inspected by each device to determine whether or not the device should react to the button code which immediately follows. (Note: do not transmit prefix codes prior to repeat codes.) If a button code is not immediately preceded by a device select prefix character, then all PMX84s in the system will react to that button code. The following table summarizes the ASCII characters to use for selecting various devices. This information is also summarized in the ASCII code chart provided at the end of this manual.

Device Select Prefix Codes

select device 1	'l'	(0x6C)
select device 2	'm'	(0x6D)
select devices 1 & 2	'n'	(0x6E)
select device 3	'o'	(0x6F)
select devices 1 & 3	'p'	(0x70)
select devices 2 & 3	'q'	(0X71)
select devices 1 & 2 & 3	'r'	(0x72)
select device 4	's'	(0x73)
select devices 1 & 4	't'	(0x74)
select devices 2 & 4	'u'	(0x75)
select devices 1 & 2 & 4	'v'	(0x76)
select devices 3 & 4	'w'	(0x77)
select devices 1 & 3 & 4	'x'	(0x78)
select devices 2 & 3 & 4	'y'	(0x79)
select devices 1 & 2 & 3 & 4	'z'	(0x7A)

Computer Control using the PMX-Link

This method of computer control is very similar to Control Button Emulation. However, by interfacing to the PMX84 using the PMX-Link, the computer is able to emulate all 200 of the button "events" which the PMX84 supports. This includes emulation of remote control activity from all four remote ports as well as from the 16 logic inputs. Each button event has a corresponding 8-bit "serial key code" (see the following table). The computer (or other controller) may emulate a button event by transmitting the corresponding 8-bit serial key code to the PMX-Link.

As mentioned earlier, the proper data communications settings are: 2400 bits per second, no parity, 1 stop bit. To avoid overrunning the PMX-Link's input buffer, the average transmission rate should not exceed 20 characters per second.

Unlike Control Button Emulation, the PMX-Link does not support device select prefix codes.

PMX-Link Serial Key Codes

button event #	serial key code (hex)	control button event
0 - 39	0x00 - 0x27	Remote Port 1 Buttons 1 - 40
40 - 79	0x28 - 0x4F	Remote Port 2 Buttons 1 - 40
80 - 119	0x50 - 0x77	Remote Port 3 Buttons 1 - 40
120 - 159	0x78 - 0x9F	Remote Port 4 Buttons 1 - 40
160	0xA0	Logic Input 1 Contact Closed
161	0xA1	Logic Input 2 Contact Closed
162	0xA2	Logic Input 3 Contact Closed
163	0xA3	Logic Input 4 Contact Closed
164	0xA4	Logic Input 1 Contact Opened
165	0xA5	Logic Input 2 Contact Opened
166	0xA6	Logic Input 3 Contact Opened
167	0xA7	Logic Input 4 Contact Opened
168	0xA8	Logic Input 5 Contact Closed
169	0xA9	Logic Input 6 Contact Closed
170	0xAA	Logic Input 7 Contact Closed
171	0xAB	Logic Input 8 Contact Closed
172	0xAC	Logic Input 5 Contact Opened
173	0xAD	Logic Input 6 Contact Opened
174	0xAE	Logic Input 7 Contact Opened
175	0xAF	Logic Input 8 Contact Opened
176	0xB0	Logic Input 9 Contact Closed
177	0xB1	Logic Input 10 Contact Closed
178	0xB2	Logic Input 11 Contact Closed
179	0xB3	Logic Input 12 Contact Closed
180	0xB4	Logic Input 9 Contact Opened
181	0xB5	Logic Input 10 Contact Opened
182	0xB6	Logic Input 11 Contact Opened
183	0xB7	Logic Input 12 Contact Opened
184	0xB8	Logic Input 13 Contact Closed
185	0xB9	Logic Input 14 Contact Closed
186	0xBA	Logic Input 15 Contact Closed
187	0xBB	Logic Input 16 Contact Closed
188	0xBC	Logic Input 13 Contact Opened
189	0xBD	Logic Input 14 Contact Opened
190	0xBE	Logic Input 15 Contact Opened
191	0xBF	Logic Input 16 Contact Opened
192	0xC0	Logic Inputs 1 - 4 All Opened
193	0xC1	Logic Inputs 5 - 8 All Opened
194	0xC2	Logic Inputs 9 - 12 All Opened
195	0xC3	Logic Inputs 13 - 16 All Opened
196	0xC4	Logic Inputs 1 - 8 All Opened
197	0xC5	Logic Inputs 9 - 16 All Opened
198	0xC6	Logic Inputs 1 - 16 All Opened
199	0xC7	Power-Up / Reset Event

Advanced Computer Control

The Advanced Computer Control command set includes commands which allow the PMX84 to return information about the system to the computer, unlike Control Button Emulation and the PMX-Link which are basically one-way control mechanisms. The following list summarizes the commands available using Advanced Computer Control, including the ASCII command character associated with each command:

!	do-macro	perform the actions for the specified button macro.
!	get-macro-definition	retrieve the definition for the specified button macro.
"	define-macro	define the specified button macro.
"	virtual-macro	perform the specified macro actions.
#	do-button	perform the actions associated with the specified button.
#	get-button-definition	retrieve the definition for the specified button.
$	define-button	define the specified button.
%	get-matrix-status	retrieve the on/off status for the entire assignment matrix.
&	set-matrix-status	set the on/off status for the entire assignment matrix.
'	do-matrix-action	perform the specified matrix assignment switch action.
(	get-logic-status	retrieve the on/off status of all logic outputs.
)	set-logic-status	set the on/off status of all logic outputs.
*****	do-logic-action	perform the specified logic output action.
+	sleep-for-10-sec.	sleep for 10 seconds, ignoring all commands.
,	read-memory	retrieve data from non-volatile configuration memory.
-	write-memory	store data in non-volatile configuration memory.
.	set-factory-defaults	resets configuration parameters to factory defaults.
/	get-version	retrieve the model information and firmware version date.

Each Advanced Computer Control command requires at least two parameter bytes (four pseudo-hex nibbles) to be sent prior to the command character. Each command will be explained in detail on the following pages.

Some of the commands cause the PMX84 to return information to the computer. For each string of information returned to the computer, the PMX84 terminates the string by transmitting the ASCII carriage return character (0x0D - represented in this document as ↵).

As mentioned earlier, the Advantage PMX84 will echo all characters it receives, regardless of whether or not the characters are valid commands or parameters. Characters greater than 0x7F are reserved and should not be transmitted by the computer. The PMX84 utilizes a subset of the standard ASCII character set. The following characters have meaning to the PMX84:

character	hexadecimal	operation
ASCII control characters	(0x00 - 0x1F)	no operation
ASCII **SPACE** character	(0x20)	no operation
! thru **/**	(0x21 - 0x2F)	Advanced Computer Control commands
0 thru **?**	(0x30 - 0x3F)	pseudo-hex parameters for Advanced Computer Control commands
@	(0x40)	Control Button Emulation Repeat Code
A	(0x41)	no operation
B thru **`**	(0x42 - 0x60)	Control Button Emulation commands (buttons 01 - 31)
a	(0x61)	no operation
b thru **j**	(0x62 - 0x6A)	Control Button Emulation commands (buttons 32 - 40)
k thru **z**	(0x6B - 0x7A)	Control Button Emulation Device Select Prefix commands
{ thru **DEL**	(0x7B - 0x7F)	no operation
0x80 thru 0xFF	(0x80 - 0xFF)	RESERVED

An ASCII code chart showing all PMX84 commands and codes is provided later in this document for your convenience. One key point to observe is that the computer may feel free to transmit spaces, tabs, carriage returns, line feeds, or any other control characters *at any time* (even between two nibbles of a pseudo-hex parameter byte) without having *any* affect on the operation of the PMX84. The PMX84 will simply echo them and then ignore them.

Device Type Bitmask and Device Number Bitmask

In a system which has more than one Advantage product connected to the computer, the Device Type Bitmask and Device Number Bitmask command parameters provide a mechanism for the computer to individually address a particular device (or a combination of devices). Every command in the Advanced Computer Control command set requires that a Device Type Bitmask and a Device Number Bitmask be transmitted as the last two parameter bytes before the computer transmits the command character itself. These two bitmask parameters bytes provide a device addressing capability to specify which of the devices in the system should execute the command. All devices which are not specifically addressed by these two bitmask values will ignore the command.

The Device Type Bitmask parameter byte supports up to eight distinct device types - one bit per device type. The eight device types are:

0x01 (bit 0) Biamp Advantage DRC 4+4 digital remote control
0x02 (bit 1) Biamp Advantage EQ28X digitally-controlled graphic equalizer
0x04 (bit 2) Biamp Advantage SPM522D stereo preamp/mixer
0x08 (bit 3) Biamp Advantage PMX84 programmable matrix switch
0x10 (bit 4) (reserved for future product)

0x20 (bit 5) (reserved for future product)
0x40 (bit 6) (reserved for future product)
0x80 (bit 7) (reserved for future product)

The Advantage PMX84 will only respond to Advanced Computer Control commands if bit 3 of the Device Type Bitmask parameter byte is a '1'. A command may be directed to more than one device type in the system by setting all of the corresponding bits in the Device Type Bitmask to '1's.

The Device Number Bitmask parameter byte supports up to eight distinct device numbers - one bit per device number. The eight device numbers are:

0x01 (bit 0) Select Device Number 1
0x02 (bit 1) Select Device Number 2
0x04 (bit 2) Select Device Number 3
0x08 (bit 3) Select Device Number 4
0x10 (bit 4) Select Device Number 5
0x20 (bit 5) Select Device Number 6
0x40 (bit 6) Select Device Number 7
0x80 (bit 7) Select Device Number 8

A particular Advantage PMX84 will only respond to Advanced Computer Control commands if the bit in the Device Number Bitmask parameter byte corresponding to its device number is a '1'. A command may be directed to more than one device number in the system by setting all of the corresponding bits in the Device Number Bitmask to '1's.

The Advanced Computer Control command set supports, in theory, up to sixty-four devices in a system - eight devices of each of the eight device types. In order for any particular device in the system to respond to an Advanced Computer Control command, the appropriate bit in both the Device Type and Device Number bitmask parameter bytes must be set to '1'.

Advanced Computer Control Data Structures

Matrix Status Data Structure

The PMX84 maintains a 4-byte data structure representing the on/off status of all 32 matrix assignment switches. Each binary bit corresponds to one assignment switch (see illustration). A '1' bit means the switch is on, and a '0' bit means the switch is off. For example, if bit 5 of the matrix[2] byte is a '1', audio input channel 6 is assigned to audio output 2.

PMX84 Matrix Status Data Structure

4-byte array with elements numbered matrix[0] thru matrix[3]

	(msb) 7	6	5	4	3	2	1	(lsb) 0
matrix[3] (transmitted first)	8→4	8→3	8→2	8→1	7→4	7→3	7→2	7→1
matrix[2]	6→4	6→3	6→2	6→1	5→4	5→3	5→2	5→1
matrix[1]	4→4	4→3	4→2	4→1	3→4	3→3	3→2	3→1
matrix[0] (transmitted last)	2→4	2→3	2→2	2→1	1→4	1→3	1→2	1→1

Logic Status Data Structure

The PMX84 maintains a 2-byte data structure representing the on/off status of all 16 logic outputs. Each binary bit corresponds to one logic output (see illustration). A '1' bit means the open collector logic output is on (a low impedance to ground), and a '0' bit means the open collector logic output is off (high impedance). For example, if bit 2 of the logic[1] byte is a '1', logic output 11 is on.

PMX84 Logic Status Data Structure

2-byte array with elements numbered logic[0] thru logic[1]

	(msb) 7	6	5	4	3	2	1	(lsb) 0
logic[1] (transmitted first)	16	15	14	13	12	11	10	9
logic[0] (transmitted last)	8	7	6	5	4	3	2	1

Button Definitions vs. Button Macros

The Advantage PMX84, like other members of the Advantage family of digitally controlled products, provides programmable or "definable" buttons. This allows the sound contractor to determine how the unit will behave for each remote control button. However, unlike the earlier Advantage products, the PMX84 firmware splits the programmable buttons into two separate parts - a "button macro", which specifies what actions are to be performed, and a "button definition" which specifies which button macro is assigned to a particular button. When the PMX84 detects that a remote control button has been pressed, it first looks-up the button definition for that button to determine which (if any) button macro is associated with that button. If a button macro has been assigned to that button, the PMX84 then looks-up the specified button macro to determine what actions to perform.

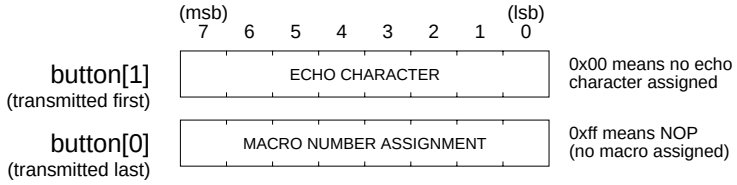
The PMX84 provides support for 200 buttons or button events (for a list of all 200 button events, see page 7). The PMX84 reserves space in its non-volatile memory for each of the 200 buttons or button events. It also allocates space for 50 button macros. A particular button macro may be assigned to more than one button - defining 4 buttons, for example, to all perform the same actions only requires 1 button macro, not 4.

Button Definition Data Structure

The PMX84 maintains data structures for 200 buttons or button events (button 0 thru button 199). Each button definition data structure consists of two bytes - one byte specifies which button macro, if any, has been assigned to the button and the other byte specifies what character, if any, should be "echoed" (transmitted) out the PMX84's serial port when that button event occurs. A macro number assignment of 0xff indicates that no macro has been assigned to the button and therefore the button is a "NOP" (it performs no operation). An echo character assignment of 0x00 means that no character will be transmitted out the serial port when that button event occurs.

PMX84 Button Definition Data Structure

2-byte array with elements numbered button[0] thru button[1]



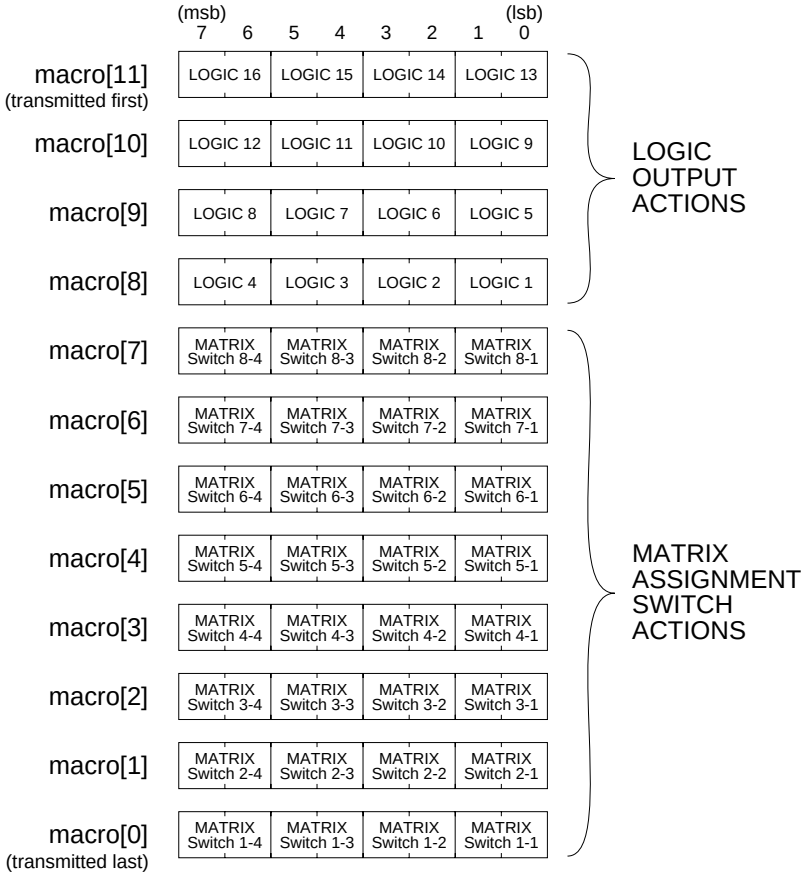
Button Macro Data Structure

The PMX84 maintains data structures for 50 button macro definitions (macro 0 thru macro 49). Each button macro data structure consists of twelve bytes - eight bytes specifying actions for the 32 matrix assignment switches and four bytes specifying actions for the 16 logic outputs. Each action code consists of two bits with the following bit patterns:

0	0	NOP (no operation)
0	1	turn OFF
1	0	turn ON
1	1	toggle

PMX84 Button Macro Data Structure

12-byte array with elements numbered macro[0] thru macro[11]



Advanced Computer Control Command Notation

For the following descriptions of the Advanced Computer Control command set, the following conventions will be used:

Each ASCII character which represents a pseudo-hex nibble will be shown in *italics*, with the following letters representing certain types of parameters:

- *a* a pseudo-hex nibble specifying an action code.
- *b* one of the pseudo-hex nibbles specifying a button data structure. Also used to as a pseudo-hex nibble to select a memory bank.
- *c* a pseudo-hex nibble specifying a checksum value.
- *d* one of the pseudo-hex nibbles in the device number bitmask which indicates which device numbers should accept the following command.
- *e* a pseudo-hex nibble specifying a memory address in the non-volatile configuration memory of the PMX84 (the ending address of a range of addresses).
- *k* a pseudo-hex nibble specifying a button serial key code.
- *l* one of the pseudo-hex nibbles specifying a logic output status data structure.
- *m* one of the pseudo-hex nibbles specifying a macro data structure or a matrix status data structure.
- *n* a pseudo-hex nibble specifying a button number or macro number.
- *o* a pseudo-hex nibble specifying a command option byte.
- *s* a pseudo-hex nibble specifying a memory address in the non-volatile configuration memory of the PMX84 (the starting address of a range of addresses).
- *x* a pseudo-hex nibble specifying a generic data value.

! do-macro

Description:

The do-macro command causes the PMX84 to look-up and perform the actions for the specified button macro.

Syntax of Command:

*nn*08*dd***!**

where:

nn	=	Macro Number + 128 (pseudo-hex)
08	=	PMX84 Device Type Bitmask
dd	=	Device Number Bitmask (pseudo-hex)
!	=	do-macro command character (0x21)

Syntax of Response:

(no response)

Example:

950820!

This example causes PMX84 number 6 to immediately look-up and perform the actions for button macro number 21.

Comments:

The decimal value 128 must be added to the Macro number. This indicates to the PMX84 that this is the do-macro form of the command.

! get-macro-definition

Description:

The get-macro-definition command causes the PMX84 to return the definition of the specified button macro. The button macro definition will be returned in the Button Macro Data Structure format.

Syntax of Command:

*nn*08*dd***!**

where:

nn	=	Macro Number (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
!	=	get-macro-definition command character (0x21)

Syntax of Response:

mmmmmmmmmmmmmmmmmmmmmmmmm↵

where:

mmmmmmmmmmmmmmmmmmmmmmmmm = Button Macro Data Structure (pseudo-hex)

Example:

command:	response:
1;0801!	**00000002555555555555555::↵**

This example causes PMX84 number 1 to retrieve its button macro definition for macro number 27 (0x1B) and return it to the computer. In this example, the macro definition was: assign input 1 to outputs 1, 2, 3, and 4; un-assign all other inputs and outputs; turn on logic output 1; leave logic outputs 2-16 the way they were.

Comments:

" define-macro

Description:

The define-macro command provides a new button macro definition for the specified macro number. The PMX84 will store this new button macro definition in its macro definition lookup table, replacing the definition that was there.

Syntax of Command:

*mmmmmmmmmmmmmmmmmmmmmmmmmmnn*08*dd*"

where:

mmmm...mmmm	=	Button Macro Data Structure (pseudo-hex)
nn	=	Macro Number (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
"	=	define-macro command character (0x22)

Syntax of Response:

(no response)

Example:

000000020000000000000002030801"

This example causes PMX84 number 1 to redefine macro number 03 to assign input 1 to output 1 and to turn on logic output number 1 and to leave all other matrix switch assignments and logic outputs the way they were.

Comments:

" virtual-macro

Description:

The virtual-macro command causes the specified button macro actions to be immediately performed. The actions are defined using the Button Macro Data Structure. The button macro definition is not stored in the PMX84's nonvolatile memory, nor is the macro definition associated with a macro number. The virtual-macro command allows the computer to specify the actions to be performed without having the PMX84 look-up an entry in its button macro definition table.

Syntax of Command:

*mmmmmmmmmmmmmmmmmmmmmmmmm*8008*dd*"

where:

mmmm...mmmm	=	Button Macro Data Structure (pseudo-hex)
80	=	command identifier
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
"	=	virtual-macro command character (0x22)

Syntax of Response:

(no response)

Example:

000000020000000000000002800801"

This example causes PMX84 number 1 to assign input 1 to output 1 and to turn on logic output number 1 and to leave all other matrix switch assignments and logic outputs the way they were.

Comments:

This command specifies actions (NOP, turn off, turn on, toggle) to be performed for all 32 matrix assignment switches and all 16 logic outputs. To perform a single matrix assignment switch or logic output action, refer to the do-matrix-action and do-logic-action commands. To set all matrix assignment switches or all logic outputs to a known state, refer to the set-matrix-status and set-logic-status commands.

do-button

Description:

The do-button command causes the PMX84 to look-up the button definition for the button (or button "event") with the specified button serial key code and, if a button macro has been assigned to that button, perform the button macro actions. The button serial key code is a number from 0 to 199 (0x00 to 0xC7).

Syntax of Command:

kk?>08*dd*#

where:

kk	=	Button Serial Key Code (pseudo-hex)
?>	=	command identifier 0xFE (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
#	=	do-button command character (0x23)

Syntax of Response:

(no response)

Example:

32?>080?#

This example causes PMX84 numbers 1, 2, 3, and 4 to look-up and perform the actions defined in their respective button definition tables for button 11 of remote port 2 (serial key code 0x32).

Comments:

get-button-definition

Description:

The get-button-definition command causes the PMX84 to return the definition of the button (or button "event") for the specified button serial key code. The button definition will be returned in the Button Definition Data Structure format. The button serial key code is a number from 0 to 199 (0x00 to 0xC7).

Syntax of Command:

*kk*08*dd*#

where:

kk	=	Button Serial Key Code (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
#	=	get-button-definition command character (0x23)

Syntax of Response:

bbbb↵

where:

bbbb	=	Button Definition Data Structure (pseudo-hex)

Example:

command:	response:
580801#	**4:07↵**

This example causes PMX84 number 1 to retrieve its button definition for button number 9 of remote port 3 (serial key code 0x58) and return it to the computer. In this example, the echo character is defined to be 'J' (0x4A), and macro number 07 is assigned to this button.

Comments:

$ define-button

Description:

The define-button command provides a new button definition for the button (or button "event") corresponding to the specified button serial key code. The PMX84 will store this new button definition in its button definition lookup table, replacing the definition that was there.

Syntax of Command:

*bbbbkk*08*dd***$**

where:

bbbb	=	Button Definition Data Structure
kk	=	Button Serial Key Code 0 - 199 (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
$	=	define-button command character (0x24)

Syntax of Response:

(no response)

Example:

4213000801$

This example provides a new button definition for button number 1 of remote port 1 (serial key code 0x00) of PMX84 number 1. The new button definition sets the echo character to 'B' (0x42) and assigns button macro number 19 (0x13) to this button.

Comments:

% get-matrix-status

Description:

The get-matrix-status command causes the PMX84 to return the on/off status of all 32 matrix assignment switches. The settings will be returned in the Matrix Status Data Structure format.

Syntax of Command:

08*dd***%**

where:

08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
%	=	get-matrix-status command character (0x25)

Syntax of Response:

mmmmmmmm↵

where:

mmmmmmmm	=	Matrix Status Data Structure (pseudo-hex)

Example:

command:	response:
0801%	**88442211**↵

This example causes PMX84 number 1 to retrieve the current status of all 32 matrix assignment switches. In this example, inputs 1 and 2 are assigned to output 1, inputs 3 and 4 are assigned to output 2, inputs 5 and 6 are assigned to output 3, and inputs 7 and 8 are assigned to output 4.

Comments:

& set-matrix-status

Description:

The set-matrix-status command establishes new settings for the on/off status of all 32 matrix assignment switches. The new settings will immediately become effective.

Syntax of Command:

*mmmmmmmm*08*dd***&**

where:

mmmmmmmm	=	Matrix Status Data Structure (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
&	=	set-matrix-status command character (0x26)

Syntax of Response:

(no response)

Example:

000084210801&

This example causes PMX84 number 1 to set its matrix assignment switches to: input 1 assigned to output 1, input 2 assigned to output 2, input 3 assigned to output 3, and input 4 assigned to output 4 (all other assignment switches off).

Comments:

This command affects all 32 matrix assignment switches. To change the status of a single matrix assignment switch, refer to the do-matrix-action command.

' do-matrix-action

Description:

The do-matrix-action command causes the PMX84 to perform the specified matrix action (NOP, turn off, turn on, toggle) for the specified matrix assignment switch.

Syntax of Command:

*aann*08*dd*'

where:

aa	=	Action code (pseudo-hex). 00 = NOP, 01 = turn off, 02 = turn on, 03 = toggle.
nn	=	Assignment Switch Number (pseudo-hex, see below)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
'	=	do-matrix-action command character (0x27)

pseudo-hex codes for assignment switch numbers (*nn*):

00	in 1 - out 1	08	in 3 - out 1	10	in 5 - out 1	18	in 7 - out 1
01	in 1 - out 2	09	in 3 - out 2	11	in 5 - out 2	19	in 7 - out 2
02	in 1 - out 3	0:	in 3 - out 3	12	in 5 - out 3	1:	in 7 - out 3
03	in 1 - out 4	0;	in 3 - out 4	13	in 5 - out 4	1;	in 7 - out 4
04	in 2 - out 1	0<	in 4 - out 1	14	in 6 - out 1	1<	in 8 - out 1
05	in 2 - out 2	0=	in 4 - out 2	15	in 6 - out 2	1=	in 8 - out 2
06	in 2 - out 3	0>	in 4 - out 3	16	in 6 - out 3	1>	in 8 - out 3
07	in 2 - out 4	0?	in 4 - out 4	17	in 6 - out 4	1?	in 8 - out 4

Syntax of Response:

(no response)

Example:

020>0801'

This example causes PMX84 number 1 to turn on the matrix assignment switch which assigns input 4 to output 3.

Comments:

(get-logic-status

Description:

The get-logic-status command causes the PMX84 to return the on/off status of all 16 logic outputs. The settings will be returned in the Logic Status Data Structure format.

Syntax of Command:

08*dd***(**

where:

08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
(	=	get-logic-status command character (0x28)

Syntax of Response:

llll↵

where:

llll	=	Logic Status Data Structure (pseudo-hex)

Example:

command:	response:
0801(	**0040**↵

This example causes PMX84 number 1 to retrieve the current status of all 16 logic outputs. In this example, logic output 7 is on and all others are off.

Comments:

) set-logic-status

Description:

The set-logic-status command establishes new settings for the on/off status of all 16 logic outputs. The new settings will immediately become effective.

Syntax of Command:

*llll*08*dd***)**

where:

llll	=	Logic Status Data Structure (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
)	=	set-logic-status command character (0x29)

Syntax of Response:

(no response)

Example:

40030801)

This example causes PMX84 number 1 to turn on logic outputs 1, 2, and 15, and turn off all other logic outputs.

Comments:

This command affects all 16 logic outputs. To change the status of a single logic output, refer to the do-logic-action command.

* do-logic-action

Description:

The do-logic-action command causes the PMX84 to perform the specified logic action (NOP, turn off, turn on, toggle) for the specified logic output.

Syntax of Command:

*aann*08*dd******

where:

aa	=	Action code (pseudo-hex). 00 = NOP, 01 = turn off, 02 = turn on, 03 = toggle.
nn	=	Logic Output Number (pseudo-hex, see below)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
*****	=	do-logic-action command character (0x2A)

pseudo-hex codes for logic output numbers (*nn*):

00	logic output 1	08	logic output 9
01	logic output 2	09	logic output 10
02	logic output 3	0:	logic output 11
03	logic output 4	0;	logic output 12
04	logic output 5	0<	logic output 13
05	logic output 6	0=	logic output 14
06	logic output 7	0>	logic output 15
07	logic output 8	0?	logic output 16

Syntax of Response:

(no response)

Example:

020<0801*

This example causes PMX84 number 1 to turn on logic output 13.

Comments:

+ sleep-for-10-seconds

Description:

The sleep-for-10-seconds command causes the PMX84 to "go to sleep" for 10 seconds, ignoring *all* serial port data communications, including Control Button commands as well as Advanced Computer Control commands. During this time, characters received by the serial port will be ignored and will not be echoed. This command was implemented to facilitate remote computer control of the PMX84 via modem (with an auto-answer modem at the PMX84). When an on-line session with a modem is finished and one modem or the other decides to disconnect or "hang up the phone", typically a spurt of unwanted spurious garbage characters occurs on the line. The PMX84 has no way of distinguishing between "garbage" characters and real characters. The last thing the computer should do before telling its modem to hang up is to issue the sleep-for-10-seconds command. This will allow plenty of time for the line to disconnect and the PMX84 will ignore all characters which it might receive during this hang-up process.

Syntax of Command:

ttdd+

where:

tt	=	Device Type Bitmask (pseudo-hex) 08 = PMX84
dd	=	Device Number Bitmask (pseudo-hex)
+	=	sleep-for-10-seconds command character (0x2B)

Syntax of Response:

(no response)

Example:

????+

This example causes *all* Advantage devices in the system to sleep for 10 seconds, ignoring all data communications.

Comments:

Note that the command character '+' is typically also the character used to return a Hayes-compatible modem to its command mode.

, read-memory

Description:

The read-memory command allows the computer to read the contents of one or more locations of the PMX84's non-volatile configuration memory. The PMX84 has 1024 bytes of configuration memory, which is arranged as four banks of 256 bytes each.

Syntax of Command:

*bbeess*08*dd***,**

where:

bb	=	memory Bank select. 00 = bank 0. 01 = bank 1. 02 = bank 2. 03 = bank 3.
ee	=	Ending memory address (pseudo-hex)
ss	=	Starting memory address (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
,	=	read-memory command character (0x2C)

Syntax of Response:

xx...(up to 256 data values)...↵

where:

xx	=	Data byte (pseudo-hex). Value in end addr is sent first.

Example:

command:
0006000801,

response:
6<000000000102↵

This example causes PMX84 number 1 to return the data values which are currently stored in locations 00 thru 06 of bank 0 of the configuration memory. In this example, memory location 00 contains the pseudo-hex value 02 and memory location 06 contains the pseudo-hex value 6<.

Comments:

To read the contents of only one location, set both the starting and ending address to the same value (the desired address). The starting address should always be less than or equal to the ending address. Valid addresses for each bank are 0x00 thru 0xFF (pseudo-hex 00 thru ??). The data is sent in reverse order (last location first).

Contents of Bank 0:

0x00 - 0x0F Global Configuration Parameters
0x10 - 0xFF Button Definitions for Buttons 0 thru 119

Contents of Bank 1:

0x00 - 0x9F Button Definitions for Buttons 120 thru 199
0xA0 - 0xFF Button Macro Definitions for Macros 0 thru 7

Contents of Bank 2:

0x00 - 0xFB Button Macro Definitions for Macros 8 thru 28
0xFC - 0xFF unused (reserved)

Contents of Bank 3:

0x00 - 0xFB Button Macro Definitions for Macros 29 thru 49
0xFC - 0xFF reserved

- write-memory

Description:

The write-memory command allows the computer to store one or more data values in the PMX84's non-volatile configuration memory beginning at a specified location. The PMX84 has 1024 bytes of configuration memory, which is arranged as four banks of 256 bytes each. Each memory-write command may include up to 16 data values to be stored in a contiguous range of memory locations for the specified memory bank. This command provides the computer with a mechanism for setting or changing the global configuration parameters for the device. It also provides the computer with a method for totally messing things up.

Syntax of Command:

*xx...(up to 16 data values)...ssoocc*08*dd*-

where:

xx..	=	up to 16 data values (pseudo-hex). The values are sent in reverse order (highest memory address first)
ss	=	starting (lowest) memory address (pseudo-hex)
oo	=	options & byte count (pseudo-hex) bits 3, 2, 1, 0 = number of data values - 1 bit 4 - reserved (must be 0) bits 6, 5 - memory Bank select. 00 = bank 0. 01 = bank 1. 10 = bank 2. 11 = bank 3. bit 7 - 1 = activate new global config params now.
cc	=	computed checksum value (pseudo-hex)
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
-	=	write-memory command character (0x2D)

Syntax of Response:

(no response)

Example:

`0302010382740801-`

This example causes PMX84 number 1 to store the values 0x01, 0x02, 0x03 in configuration memory bank 0 beginning at address 0x03 and causes the unit to retrieve and activate its global configuration parameters.

Comments:

In order to insure data integrity, a checksum value is included with this command. The checksum is computed by performing a simple 8-bit (modulo 256) sum of all of the data values plus the starting address value plus the option byte value and then performing a 1's complement of the sum. Note that the checksum calculation is performed on the actual 8-bit binary values, before any values are converted to pseudo-hex format for transmission. The PMX84 will perform the same calculation on the data values which it receives and compare its computed checksum to the received checksum. If the two values do not match, the PMX84 will ignore the entire command.

The PMX84 normally retrieves and activates its global configuration parameters from the non-volatile configuration memory only during its power-up sequence. If you use the write-memory command to change a global configuration parameter (which is why the command was provided), the PMX84 has to specifically be told to re-load and activate its configuration parameters after the new value is stored in its memory. Bit 7 of the options & byte count parameter may be set to tell the unit to automatically retrieve and activate its configuration parameters after the memory write operation is complete.

. set-factory-defaults

Description:

The set-factory-defaults command may be used to force the PMX84 to reset some or all of its configuration information to the factory default settings. The first two parameters for this command (< and >) are dummy parameters which were implemented to help prevent an accidental restoration of the factory defaults due to an error in data transmission.

Syntax of Command:

<>*oo***08***dd***.**

where:

<	=	the ASCII character < (0x3C)
>	=	the ASCII character > (0x3E)
oo	=	Option byte (pseudo-hex): bit 0 - 1 = set Button Definition defaults bit 1 - 1 = set Button Macro Definition defaults bit 2 - 1 = set Global Config Parameter defaults bit 7 - 1 = activate new Global Config Params now
08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
.	=	set-factory-defaults command character (0x2E)

Syntax of Response:

(no response)

Example:

<>84080?.

This example causes PMX84 numbers 1, 2, 3, and 4 to restore their global configuration parameters to the factory default settings and to retrieve and activate those new settings.

Comments:

/ get-version

Description:

The get-version command causes the PMX84 to return its model identification code and firmware version to the computer. The firmware version number is simply the release date of the firmware, in a slightly modified standard American format of *mm***:***dd***:***yy*. These values are decimal digits, not pseudo-hex notation. For example, February 12, 1996 would be represented as **02:12:96**. The colon character (**:**) is used as a separator instead of the more conventional slash character, since the slash character is used as a computer command character by the PMX84.

Syntax of Command:

08*dd*/

where:

08	=	Device Type Bitmask (pseudo-hex)
dd	=	Device Number Bitmask (pseudo-hex)
/	=	get-version command character (0x2F)

Syntax of Response:

02 *mm***:***dd***:***yy*↵

where:

02	=	initial model i.d. (0x30 followed by 0x32)
	=	ASCII space character (0x20)
mm	=	2-digit decimal month number
:	=	ASCII character **:** (0x3A)
dd	=	2-digit decimal day of the month
:	=	ASCII character **:** (0x3A)
yy	=	2-digit decimal year number

Example:

command:	response:
0801/	**02 02:12:96**↵

This example causes PMX84 number 1 to return its model identification code and firmware version.

Comments:

Advanced Computer Command Summary

*nn*08*dd***!**	do-macro
*nn*08*dd***!**	get-macro-definition
*mmmmmmmmmmmmmmmmmmmmmmmmmmnn*08*dd***"**	define-macro
*mmmmmmmmmmmmmmmmmmmmmmmmmmm*8008*dd***"**	virtual-macro
kk?>08*dd***#**	do-button
*kk*08*dd***#**	get-button definition
*bbbbkk*08*dd***$**	define-button
08*dd***%**	get-matrix-status
*mmmmmmmm*08*dd***&**	set-matrix-status
*aann*08*dd***'**	do-matrix-action
08*dd***(**	get-logic-status
*llll*08*dd***)**	set-logic-status
*aann*08*dd******	do-logic-action
08*dd***+**	sleep-for-10-seconds
*bbeess*08*dd***,**	read-memory
*....xxssoocc*08*dd***-**	write-memory
<>*oo*08*dd***.**	set-factory-defaults
08*dd***/**	get-version

- *a* a pseudo-hex nibble specifying an action code.
- *b* one of the pseudo-hex nibbles specifying a button data structure. Also used to as a pseudo-hex nibble to select a memory bank.
- *c* a pseudo-hex nibble specifying a checksum value.
- *d* one of the pseudo-hex nibbles in the device number bitmask which indicates which device numbers should accept the following command.
- *e* a pseudo-hex nibble specifying a memory address in the non-volatile configuration memory of the PMX84 (the ending address of a range of addresses).
- *k* a pseudo-hex nibble specifying a button serial key code.
- *l* one of the pseudo-hex nibbles specifying a logic output status data structure.
- *m* one of the pseudo-hex nibbles specifying a macro data structure or a matrix status data structure.
- *n* a pseudo-hex nibble specifying a button number or macro number.
- *o* a pseudo-hex nibble specifying a command option byte.
- *s* a pseudo-hex nibble specifying a memory address in the non-volatile configuration memory of the PMX84 (the starting address of a range of addresses).
- *x* a pseudo-hex nibble specifying a generic data value.

.

ASCII Code Chart

with Decimal & Hexadecimal Equivalents and Advantage PMX84 Commands

000. 0x00 **NUL**	016. 0x10 **DLE**	032. 0x20 **(space)**	048. 0x30 **0** nibble 0x0	064. 0x40 **@** repeat code	080. 0x50 **P** button 15	096. 0x60 **`** button 31	112. 0x70 **p** select 1,3
001. 0x01 **SOH**	017. 0x11 **DC1**	033. 0x21 **!** do/get-macro	049. 0x31 **1** nibble 0x1	065. 0x41 **A**	081. 0x51 **Q** button 16	097. 0x61 **a**	113. 0x71 **q** select 2,3
002. 0x02 **STX**	018. 0x12 **DC2**	034. 0x22 **"** def/virt-macro	050. 0x32 **2** nibble 0x2	066. 0x42 **B** button 01	082. 0x52 **R** button 17	098. 0x62 **b** button 32	114. 0x72 **r** select 1,2,3
003. 0x03 **ETX**	019. 0x13 **DC3**	035. 0x23 **#** do/get-button	051. 0x33 **3** nibble 0x3	067. 0x43 **C** button 02	083. 0x53 **S** button 18	099. 0x63 **c** button 33	115. 0x73 **s** select 4
004. 0x04 **EOT**	020. 0x14 **DC4**	036. 0x24 **$** define button	052. 0x34 **4** nibble 0x4	068. 0x44 **D** button 03	084. 0x54 **T** button 19	100. 0x64 **d** button 34	116. 0x74 **t** select 1,4
005. 0x05 **ENQ**	021. 0x15 **NAK**	037. 0x25 **%** get-matrix	053. 0x35 **5** nibble 0x5	069. 0x45 **E** button 04	085. 0x55 **U** button 20	101. 0x65 **e** button 35	117. 0x75 **u** select 2,4
006. 0x06 **ACK**	022. 0x16 **SYN**	038. 0x26 **&** set-matrix	054. 0x36 **6** nibble 0x6	070. 0x46 **F** button 05	086. 0x56 **V** button 21	102. 0x66 **f** button 36	118. 0x76 **v** select 1,2,4
007. 0x07 **BEL**	023. 0x17 **ETB**	039. 0x27 **'** do-matrix	055. 0x37 **7** nibble 0x7	071. 0x47 **G** button 06	087. 0x57 **W** button 22	103. 0x67 **g** button 37	119. 0x77 **w** select 3,4
008. 0x08 **BS**	024. 0x18 **CAN**	040. 0x28 **(** get-logic	056. 0x38 **8** nibble 0x8	072. 0x48 **H** button 07	088. 0x58 **X** button 23	104. 0x68 **h** button 38	120. 0x78 **x** select 1,3,4
009. 0x09 **HT**	025. 0x19 **EM**	041. 0x29 **)** set-logic	057. 0x39 **9** nibble 0x9	073. 0x49 **I** button 08	089. 0x59 **Y** button 24	105. 0x69 **i** button 39	121. 0x79 **y** select 2,3,4
010. 0x0A **LF**	026. 0x1A **SUB**	042. 0x2A ***** do-logic	058. 0x3A **:** nibble 0xA	074. 0x4A **J** button 09	090. 0x5A **Z** button 25	106. 0x6A **j** button 40	122. 0x7A **z** select 1,2,3,4
011. 0x0B **VT**	027. 0x1B **ESC**	043. 0x2B **+** sleep 10 sec.	059. 0x3B **;** nibble 0xB	075. 0x4B **K** button 10	091. 0x5B **[** button 26	107. 0x6B **k** select none	123. 0x7B **{**
012. 0x0C **FF**	028. 0x1C **FS**	044. 0x2C **,** read memory	060. 0x3C **<** nibble 0xC	076. 0x4C **L** button 11	092. 0x5C **** button 27	108. 0x6C **l** select 1	124. 0x7C **|**
013. 0x0D **CR**	029. 0x1D **GS**	045. 0x2D **-** write memory	061. 0x3D **=** nibble 0xD	077. 0x4D **M** button 12	093. 0x5D **]** button 28	109. 0x6D **m** select 2	125. 0x7D **}**
014. 0x0E **SO**	030. 0x1E **RS**	046. 0x2E **.** set defaults	062. 0x3E **>** nibble 0xE	078. 0x4E **N** button 13	094. 0x5E **^** button 29	110. 0x6E **n** select 1,2	126. 0x7E **~**
015. 0x0F **SI**	031. 0x1F **US**	047. 0x2F **/** get version	063. 0x3F **?** nibble 0xF	079. 0x4F **O** button 14	095. 0x5F **_** button 30	111. 0x6F **o** select 3	127. 0x7F **DEL**

HEXADECIMAL CONVERSION CHART

binary	decimal	hex	pseudo
0000 0000	0.	0x00	00
0000 0001	1.	0x01	01
0000 0010	2.	0x02	02
0000 0011	3.	0x03	03
0000 0100	4.	0x04	04
0000 0101	5.	0x05	05
0000 0110	6.	0x06	06
0000 0111	7.	0x07	07
0000 1000	8.	0x08	08
0000 1001	9.	0x09	09
0000 1010	10.	0x0a	0:
0000 1011	11.	0x0b	0;
0000 1100	12.	0x0c	0<
0000 1101	13.	0x0d	0=
0000 1110	14.	0x0e	0>
0000 1111	15.	0x0f	0?
0001 0000	16.	0x10	10
0001 0001	17.	0x11	11
0001 0010	18.	0x12	12
0001 0011	19.	0x13	13
0001 0100	20.	0x14	14
0001 0101	21.	0x15	15
0001 0110	22.	0x16	16
0001 0111	23.	0x17	17
0001 1000	24.	0x18	18
0001 1001	25.	0x19	19
0001 1010	26.	0x1a	1:
0001 1011	27.	0x1b	1;
0001 1100	28.	0x1c	1<
0001 1101	29.	0x1d	1=
0001 1110	30.	0x1e	1>
0001 1111	31.	0x1f	1?
0010 0000	32.	0x20	20
0010 0001	33.	0x21	21
0010 0010	34.	0x22	22
0010 0011	35.	0x23	23
0010 0100	36.	0x24	24
0010 0101	37.	0x25	25
0010 0110	38.	0x26	26
0010 0111	39.	0x27	27
0010 1000	40.	0x28	28
0010 1001	41.	0x29	29
0010 1010	42.	0x2a	2:
0010 1011	43.	0x2b	2;
0010 1100	44.	0x2c	2<
0010 1101	45.	0x2d	2=
0010 1110	46.	0x2e	2>
0010 1111	47.	0x2f	2?
0011 0000	48.	0x30	30
0011 0001	49.	0x31	31
0011 0010	50.	0x32	32
0011 0011	51.	0x33	33
0011 0100	52.	0x34	34
0011 0101	53.	0x35	35
0011 0110	54.	0x36	36
0011 0111	55.	0x37	37
0011 1000	56.	0x38	38
0011 1001	57.	0x39	39
0011 1010	58.	0x3a	3:
0011 1011	59.	0x3b	3;
0011 1100	60.	0x3c	3<
0011 1101	61.	0x3d	3=
0011 1110	62.	0x3e	3>
0011 1111	63.	0x3f	3?
0100 0000	64.	0x40	40
0100 0001	65.	0x41	41
0100 0010	66.	0x42	42
0100 0011	67.	0x43	43
0100 0100	68.	0x44	44
0100 0101	69.	0x45	45
0100 0110	70.	0x46	46
0100 0111	71.	0x47	47
0100 1000	72.	0x48	48
0100 1001	73.	0x49	49
0100 1010	74.	0x4a	4:
0100 1011	75.	0x4b	4;
0100 1100	76.	0x4c	4<
0100 1101	77.	0x4d	4=
0100 1110	78.	0x4e	4>
0100 1111	79.	0x4f	4?
0101 0000	80.	0x50	50
0101 0001	81.	0x51	51
0101 0010	82.	0x52	52
0101 0011	83.	0x53	53
0101 0100	84.	0x54	54
0101 0101	85.	0x55	55
0101 0110	86.	0x56	56
0101 0111	87.	0x57	57
0101 1000	88.	0x58	58
0101 1001	89.	0x59	59
0101 1010	90.	0x5a	5:
0101 1011	91.	0x5b	5;
0101 1100	92.	0x5c	5<
0101 1101	93.	0x5d	5=
0101 1110	94.	0x5e	5>
0101 1111	95.	0x5f	5?
0110 0000	96.	0x60	60
0110 0001	97.	0x61	61
0110 0010	98.	0x62	62
0110 0011	99.	0x63	63
0110 0100	100.	0x64	64
0110 0101	101.	0x65	65
0110 0110	102.	0x66	66
0110 0111	103.	0x67	67
0110 1000	104.	0x68	68
0110 1001	105.	0x69	69
0110 1010	106.	0x6a	6:
0110 1011	107.	0x6b	6;
0110 1100	108.	0x6c	6<
0110 1101	109.	0x6d	6=
0110 1110	110.	0x6e	6>
0110 1111	111.	0x6f	6?
0111 0000	112.	0x70	70
0111 0001	113.	0x71	71
0111 0010	114.	0x72	72
0111 0011	115.	0x73	73
0111 0100	116.	0x74	74
0111 0101	117.	0x75	75
0111 0110	118.	0x76	76
0111 0111	119.	0x77	77
0111 1000	120.	0x78	78
0111 1001	121.	0x79	79
0111 1010	122.	0x7a	7:
0111 1011	123.	0x7b	7;
0111 1100	124.	0x7c	7<
0111 1101	125.	0x7d	7=
0111 1110	126.	0x7e	7>
0111 1111	127.	0x7f	7?
1000 0000	128.	0x80	80
1000 0001	129.	0x81	81
1000 0010	130.	0x82	82
1000 0011	131.	0x83	83
1000 0100	132.	0x84	84
1000 0101	133.	0x85	85
1000 0110	134.	0x86	86
1000 0111	135.	0x87	87
1000 1000	136.	0x88	88
1000 1001	137.	0x89	89
1000 1010	138.	0x8a	8:
1000 1011	139.	0x8b	8;
1000 1100	140.	0x8c	8<
1000 1101	141.	0x8d	8=
1000 1110	142.	0x8e	8>
1000 1111	143.	0x8f	8?
1001 0000	144.	0x90	90
1001 0001	145.	0x91	91
1001 0010	146.	0x92	92
1001 0011	147.	0x93	93
1001 0100	148.	0x94	94
1001 0101	149.	0x95	95
1001 0110	150.	0x96	96
1001 0111	151.	0x97	97
1001 1000	152.	0x98	98
1001 1001	153.	0x99	99
1001 1010	154.	0x9a	9:
1001 1011	155.	0x9b	9;
1001 1100	156.	0x9c	9<
1001 1101	157.	0x9d	9=
1001 1110	158.	0x9e	9>
1001 1111	159.	0x9f	9?
1010 0000	160.	0xa0	:0
1010 0001	161.	0xa1	:1
1010 0010	162.	0xa2	:2
1010 0011	163.	0xa3	:3
1010 0100	164.	0xa4	:4
1010 0101	165.	0xa5	:5
1010 0110	166.	0xa6	:6
1010 0111	167.	0xa7	:7
1010 1000	168.	0xa8	:8
1010 1001	169.	0xa9	:9
1010 1010	170.	0xaa	::
1010 1011	171.	0xab	:;
1010 1100	172.	0xac	:<
1010 1101	173.	0xad	:=
1010 1110	174.	0xae	:>
1010 1111	175.	0xaf	:?
1011 0000	176.	0xb0	;0
1011 0001	177.	0xb1	;1
1011 0010	178.	0xb2	;2
1011 0011	179.	0xb3	;3
1011 0100	180.	0xb4	;4
1011 0101	181.	0xb5	;5
1011 0110	182.	0xb6	;6
1011 0111	183.	0xb7	;7
1011 1000	184.	0xb8	;8
1011 1001	185.	0xb9	;9
1011 1010	186.	0xba	;:
1011 1011	187.	0xbb	;;
1011 1100	188.	0xbc	;<
1011 1101	189.	0xbd	;=
1011 1110	190.	0xbe	;>
1011 1111	191.	0xbf	;?
1100 0000	192.	0xc0	<0
1100 0001	193.	0xc1	<1
1100 0010	194.	0xc2	<2
1100 0011	195.	0xc3	<3
1100 0100	196.	0xc4	<4
1100 0101	197.	0xc5	<5
1100 0110	198.	0xc6	<6
1100 0111	199.	0xc7	<7
1100 1000	200.	0xc8	<8
1100 1001	201.	0xc9	<9
1100 1010	202.	0xca	<:
1100 1011	203.	0xcb	<;
1100 1100	204.	0xcc	<<
1100 1101	205.	0xcd	<=
1100 1110	206.	0xce	<>
1100 1111	207.	0xcf	<?
1101 0000	208.	0xd0	=0
1101 0001	209.	0xd1	=1
1101 0010	210.	0xd2	=2
1101 0011	211.	0xd3	=3
1101 0100	212.	0xd4	=4
1101 0101	213.	0xd5	=5
1101 0110	214.	0xd6	=6
1101 0111	215.	0xd7	=7
1101 1000	216.	0xd8	=8
1101 1001	217.	0xd9	=9
1101 1010	218.	0xda	=:
1101 1011	219.	0xdb	=;
1101 1100	220.	0xdc	=<
1101 1101	221.	0xdd	==
1101 1110	222.	0xde	=>
1101 1111	223.	0xdf	=?
1110 0000	224.	0xe0	>0
1110 0001	225.	0xe1	>1
1110 0010	226.	0xe2	>2
1110 0011	227.	0xe3	>3
1110 0100	228.	0xe4	>4
1110 0101	229.	0xe5	>5
1110 0110	230.	0xe6	>6
1110 0111	231.	0xe7	>7
1110 1000	232.	0xe8	>8
1110 1001	233.	0xe9	>9
1110 1010	234.	0xea	>:
1110 1011	235.	0xeb	>;
1110 1100	236.	0xec	><
1110 1101	237.	0xed	>=
1110 1110	238.	0xee	>>
1110 1111	239.	0xef	>?
1111 0000	240.	0xf0	?0
1111 0001	241.	0xf1	?1
1111 0010	242.	0xf2	?2
1111 0011	243.	0xf3	?3
1111 0100	244.	0xf4	?4
1111 0101	245.	0xf5	?5
1111 0110	246.	0xf6	?6
1111 0111	247.	0xf7	?7
1111 1000	248.	0xf8	?8
1111 1001	249.	0xf9	?9
1111 1010	250.	0xfa	?:
1111 1011	251.	0xfb	?;
1111 1100	252.	0xfc	?<
1111 1101	253.	0xfd	?=
1111 1110	254.	0xfe	?>
1111 1111	255.	0xff	??