Tel : +44 (0)1798 877000
hvsales@spellmanhv.co.uk

Broomers Park, Pulborough,
W. Sussex, UK. RH20 2RY

Product:

MPD Series

Title:

RS232 & RS485 Serial Communications Protocol

Document Number:

48113-21

Issue:

3

Date:

12/April/2023

Change History

ISSUE	DATE	NAME	SECTION	CHANGE
A			All	Created
B	03/05/2019		3.1	Added 15kV & 20kV ID information
C	21/10/2020		3.1	Added 30kV ID information
D	15/02/2021		3.2	Added Baud rate and Wobbler commands
1	04/05/2021	JS	All	New Template
2	26/04/2022	JS+CC	All 3 3.2	Template update Notes added about ASCII control characters Added Response Time (RT) command. Notes and units of measure added for clarity
3	12/04/2023	JS	1 3.1	Removed “master/slave” terminology MPD1 (1kV) device type added

Contents

1 Introduction

The MPD Series protocol is based on the general protocol used with Spellman HV Electronic Ltd power supplies which is described in this document. At the hardware level the protocol runs over an RS232/RS485 two wire interface. Up to 99 units can be connected to any one network.

2 Hardware

The default data rate is 8-bit, no parity at 9600 baud with 1 stop bit.

3 General Agreements

All commands are encoded using ASCII.

All commands sent to the units will receive a response except when the 'Call All' address of "00" is used.

All commands take the general form shown below, which includes both standard ASCII and ASCII control characters.

<STX><ADDR><DEVTYPE><CMD><OPERATOR>< DATA><CSUM><LF>

Element	Comment
<STX>	Single unprintable ASCII control character 2, 0x02. Start of text
<ADDR>	Two ASCII characters representing the decimal address of the unit to which the message is being sent. *Valid range: "00" → "99"* *Note:* *All units are shipped with the default address "01".* *Address "00" is reserved for use as a broadcast address. This makes it possible to send the same command to ALL connected modules.* *When address "00" is used, the connected modules will NOT respond unless command sent is "*ID?*".*
<DEVTYPE>	Two ASCII characters representing the type of unit *For more information see paragraph 3.1*
<CMD>	Two ASCII characters that define the command. The <CMD> is sent in both the command from the host and the response from the MPD module. *For more information see paragraph 3.2*
<OPERATOR>	Optional single ASCII character that, when present, indicates the operation to be performed. *'?'* *Is used to request the current value of a module parameter.* *'='* *Is used to set or program a module parameter.* *'*'* *Is used by the only the unit to respond the host when a command is invalid. For more information see paragraph 3.2*
<DATA>	Optional command argument, up to eight ASCII characters.
<CSUM>	Two ASCII characters representing the hexadecimal value of checksum. *In the event the <CSUM> received is invalid, the command cannot be trusted and so there will not be a response from the unit.* *See Appendix 2 for more information on checksum calculation.*
<LF>	Single unprintable ASCII control character 10, 0x0A. Line Feed

Spellman High Voltage Electronics Limited | +44 (0)1798 877000 | hvsales@spellmanhv.co.uk | Broomers Park, Pulborough, W. Sussex, UK. RH20 2RY

3.1 Device Types for the MPD Series

Model	Output (kV)	Power (Watts)	<DEVTYPE> *Note: Not in order*
MPD1	1	10	“01” (0x30, 0x31)
-	-		“02” (0x30, 0x32)
-	-		“03” (0x30, 0x33)
MPD2.5	2.5		“10” (0x31, 0x30)
-	-		“04” (0x30, 0x34)
MPD5	5		“05” (0x30, 0x35)
MPD10	10		“06” (0x30, 0x36)
MPD15	15		“07” (0x30, 0x37)
MPD20	20		“08” (0x30, 0x38)
MPD30	30		“09” (0x30, 0x39)

Note: *In time, more models may be added.*

3.2 Command Summary

<table>
<tr><th><CMD>
<OPERATOR>
<DATA></th><th>Description</th><th>Response</th><th>Comment</th></tr>
<tr><td>CF=1</td><td>Clear faults</td><td></td><td></td></tr>
<tr><td>EN?</td><td>Read enable state</td><td rowspan="2">EN=x</td><td rowspan="2">Where x 0 Disable
1 Enable</td></tr>
<tr><td>EN=x</td><td>Set enable state</td></tr>
<tr><td>I1?</td><td>Read present value of current limit</td><td rowspan="2">I1=xxxxx.x</td><td rowspan="2">Where xxxxx.x is the current limit value in µA
0 µA →Maximum current limit of unit.</td></tr>
<tr><td>I1=xxxxx.x</td><td>Set current limit</td></tr>
<tr><td>ID?</td><td>Read unit address</td><td rowspan="2">ID=xx</td><td rowspan="2">Where xx is the unit address (01→99)
Note, in order to set the unit address:
i) Ensure only one unit is connected to the host.
ii) Use <ADDR> = “00”</td></tr>
<tr><td>ID=xx</td><td>Set unit address</td></tr>
<tr><td>M0?</td><td>Read voltage monitor</td><td>M0=xxxxx.x</td><td>Where xxxxx.x is the voltage value in V
0V → Maximum voltage of unit.</td></tr>
<tr><td>M1?</td><td>Read current monitor</td><td>M1=xxxxx.x</td><td>Where xxxxx.x is the current value in µA
0µA → Maximum current of unit.</td></tr>
<tr><td>R0?</td><td>Read voltage monitor raw value</td><td>R0=XXXX</td><td>Where XXXX is ASCII hexadecimal value.
Range = 0x0000→ 0xFFFF</td></tr>
<tr><td>R1?</td><td>Read current monitor raw value</td><td>R1=XXXX</td><td>Where XXXX is ASCII hexadecimal value.
Range = 0x0000 → 0xFFFF</td></tr>
<tr><td>SN?</td><td>Request unit firmware ID number</td><td>SN=48113-14</td><td>This number is used by Spellman High Voltage to identify the module’s firmware.</td></tr>
<tr><td>SR?</td><td>Read unit status register</td><td>SR=XXXX</td><td>Where XXXX is ASCII hexadecimal value.
Range = 0x0000→ 0x00FF
The least significant byte shows the status register
<table>
<tr><th>Bit</th><th>Meaning</th><th>Bit value</th></tr>
<tr><td>0</td><td>Enabled</td><td>1 = Enabled</td></tr>
<tr><td>1</td><td>Fault</td><td>1 = Fault</td></tr>
<tr><td>2</td><td>Over voltage</td><td>1 = Over voltage</td></tr>
<tr><td>3</td><td>Over current</td><td>1 = Over current</td></tr>
<tr><td>4</td><td>Over temperature</td><td>1 = Over temperature</td></tr>
<tr><td>5</td><td>Supply Rail</td><td>1 = supply out of range (< 19V or >26.5V)</td></tr>
<tr><td>6</td><td>Hardware Enable</td><td>1 = Enabled through pin</td></tr>
<tr><td>7</td><td>Software Enable</td><td>1 = Enabled through software</td></tr>
</table></td></tr>
<tr><td>SW?</td><td>Read firmware version number</td><td>SW=Vx.yy</td><td>Where x Major version number
yy Minor revision number</td></tr>
<tr><td>V1?</td><td>Read present value of output voltage</td><td rowspan="2">V1=xxxxx.x</td><td rowspan="2">Where xxxxx.x is the voltage amplitude in V
0V → Maximum voltage of unit.</td></tr>
<tr><td>V1=xxxxx.x</td><td>Set output voltage</td></tr>
<tr><td>BD=x</td><td>Set serial baud rate</td><td>No reply</td><td>Where x 0 9600 baud (default)
1 19200 baud
2 115200 baud</td></tr>
<tr><td>WS?</td><td>Read wobbler state</td><td rowspan="2">WS=x</td><td rowspan="2">Where x 0 Wobbler Disable
1 Wobbler Enable</td></tr>
<tr><td>WS=x</td><td>Set wobbler state</td></tr>
<tr><td>WC?</td><td>Read Wobbler Period</td><td rowspan="2">WC=xxxx</td><td rowspan="2">Where xxxx is the period in ms
100ms → 2seconds</td></tr>
<tr><td>WC=xxxx</td><td>Set Wobbler Period</td></tr>
<tr><td>WV?</td><td>Read Wobbler Amplitude</td><td rowspan="2">WV=xxx</td><td rowspan="2">Where xxx is the amplitude in V
1V → 300V</td></tr>
<tr><td>WV=xxx</td><td>Set Wobbler Amplitude</td></tr>
<tr><td>RT?</td><td>Read Response Time Delay</td><td rowspan="2">RT=xxxx</td><td rowspan="2">Delays the response to a command or request by a specified time.

Where xxxx =four-digit hexadecimal number representing the response delay.

xxxx = hex(Tdelay[µs]/10)

Valid Range for Tdelay in µs is:
0 µS to turn the response delay off (fastest reply) or 100 µs to 2000 µs

Example: For a 150µs delay the command sent will be RT=0x000F

Note:
At Issue 2 of this document, the RT command is ONLY implemented on the 10Watt version of the MPD.</td></tr>
<tr><td>RT=xxxx</td><td>Set Response Time Delay</td></tr>
</table>

Appendix 1 - Example Command / Response

Example 1 – Set Voltage

Sets the voltage demand of an MPD2.5 at address “01” to 2.5kV.

Command: *(Host→Unit)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<DATA>							<CSUM>		<LF>
Command (ASCII Characters)	<STX>	“01”		“10”		“V1”		“=”	“02500.0”							“65”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	3D	30	32	35	30	30	2E	30	36	35	0A

Response: *(Unit→Host)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<DATA>							<CSUM>		<LF>
Response (ASCII Characters)	<STX>	“01”		“10”		“V1”		“=”	“02500.0”							“65”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	3D	30	32	35	30	30	2E	30	36	35	0A

The response confirms that the voltage demand was set to 2.5kV.

Example 2 – Read Voltage

Reads the voltage demand of an MPD2.5 at address “01”.

Command: *(Host→Unit)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<CSUM>		<LF>
Command (ASCII Characters)	<STX>	“01”		“10”		“V1”		“?”	“78”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	3F	37	38	0A

Note: *The optional <DATA> element is not included*

Response: *(Unit→Host)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<DATA>							<CSUM>		<LF>
Response (ASCII Characters)	<STX>	“01”		“10”		“V1”		“=”	“01000.0”							“6B”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	3D	30	31	30	30	30	2E	30	36	42	0A

The response indicates that the voltage demand is set to 1kV.

Example 3 – Invalid Command

Illustrates a Host←→Unit exchange where an invalid command is sent by the host. In this instance, a ‘NAK’ is issued by the unit.

Command: *(Host→Unit)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<CSUM>		<LF>
Command (ASCII Characters)	<STX>	“01”		“10”		“V1”		“!”	“56”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	21	35	36	0A

Note: *The <CSUM> above is invalid.*

Response: *(Unit→Host)*

ELEMENT	<STX>	<ADDR>		<DEVTYPE>		<CMD>		<OPERATOR>	<CSUM>		<LF>
Response (ASCII Characters)	<STX>	“01”		“10”		“V1”		“*”	“4D”		<LF>
ASCII Value (Hex) for CSUM calculation	02	30	31	31	30	56	31	2A	34	44	0A

The response indicates that the above command was invalid.

Appendix 2 - <CSUM> Checksum calculation

The checksum is calculated as follows:

- Sum the ASCII values for the characters in the <ADDR>, <DEVTYPE>, <CMD>, <OPERATOR> and <DATA> elements into a 16-bit (or larger) word. The ASCII values are added as unsigned integers.
- Subtract the sum from 512 (0x200).
- Truncate the result down to the eight least significant bits.
- Clear the most significant bit (bit 7) of the resultant byte, (bitwise AND with 0x7F).
- Set the next most significant bit (bit 6) of the resultant byte (bitwise OR with 0x40).

Using this method, the checksum is always a number between 0x40 and 0x7F. The checksum can never be confused with the <STX> or <LF>, since these have non-overlapping ASCII values.

Visual Basic .NET checksum calculation example

The following “VB.NET” function below shows how to calculate the <CSUM> element.

The example below gives the example of a message to request the status register value.

It assumes a unit with an <ADDR> = “01” and <DEVTYPE> = “06”. The resultant checksum is 0x55.

```
CRC = CheckSum("0106SR?")       'CRC will be set to 0x55

Function CheckSum(ByVal message As String) As Int16
    Dim I As Integer
    Dim CkSum As UInt16 = 0

    If Len(message) <= 0 Then
        Return 0
        Exit Function
    End If

    For I = 0 To Len(message) - 1
        CkSum += Asc(message.Substring(I, 1))
    Next I
    CkSum = &H200 - CkSum
    CkSum = CkSum And &H7F
    CkSum = CkSum Or &H40

    Return CkSum

End Function
```

C checksum calculation example

The example below is C function that also calculates the <CSUM> element.

```
#define CHECKSUM_FAIL_VALUE_OUTSIDE_ACCEPTABLE_RANGE    0xFFFE
#define CHECKSUM_NEGATION                               0x200
#define LOWER_CHECKSUM_BOUND                            0x40
#define UPPER_CHECKSUM_BOUND                            0x7F

//=============================================================================
// Function: uart_CalculateChecksum
// Args    : unsigned char * message        - pointer to starting character
//           WORD index_of_last_character   - index of last character
// Returns : WORD                           - calculated checksum value
// Notes   : calculate checksum from bytes
//=============================================================================
WORD uart_CalculateChecksum(unsigned char * message, WORD index_of_last_character)
{
    WORD loop_total = 0;
    WORD checksum = 0;
    WORD loop_count;
    BYTE c;

    for (loop_count = 0; loop_count <= index_of_last_character; loop_count++)
    {
        c = message[loop_count];
        loop_total += message[loop_count];
    }
    checksum = CHECKSUM_NEGATION - loop_total;
    checksum &= UPPER_CHECKSUM_BOUND;
    checksum |= LOWER_CHECKSUM_BOUND;

    if (checksum < LOWER_CHECKSUM_BOUND || checksum > UPPER_CHECKSUM_BOUND)
    {
        return CHECKSUM_FAIL_VALUE_OUTSIDE_ACCEPTABLE_RANGE;
    }
    return checksum;
}
```