RS-232C Serial Interface Documentation

Power Supply – Models: 1785B, 1786B, 1787B, 1788

Packet Structure

The DC Load is programmed using packets of bytes. A packet always contains 26 bytes in hexadecimal, either going to or coming from the instrument. The basic programming rule is:

> You send a **26 byte** packet to the instrument. You then read a **26 byte** packet back from the DC Load to either
>
> - Get the status of your submitted packet, or
> - Get the data you requested.

The following are conventions to follow:

1. Hexadecimal integers will be represented by the prefix 0x.
2. Numbers are in base 10 number system unless otherwise indicated.
3. Byte numbering is zero-based, meaning numbering starts with 0.
4. Some commands are sent in little-endian format. Refer to example for details.

The structure of each 26 byte packet is:

Byte 0	Byte 1	Byte 2	Byte 3 to 24	Byte 25
0xAA	Address	Command	Command's data	Checksum

- **0xAA** is ALWAYS the first byte of any command packet or returned packet.
- **Address** must be a byte that is between 0x00 and 0xFE. Setting of the address is optional. It is not required to communicate with the instrument. The address can be set from the front panel and is stored in non-volatile memory. This feature is useful when communicating via USB, and connecting several instruments, e.g. via a USB hub. In this scenario, Windows assigns a virtual COM port to each device which is unknown prior to establishing communications with the instrument (could be different each time). In this case, the user can correlate each virtual COM port randomly assigned by Windows with a user defined address.
- **Command** is a byte that identifies which DC Load command is used.

- **Command's data** contains parameter information for the command or the data that is requested via a previous command. Some commands have no data at all. It is a good programming practice to set all unused bytes to 0x00.
- **Checksum** is the number of the arithmetic sum of each of the bytes modulo 256. Basically, the sum of byte 0 to byte 25 equals the value of byte 26. In some cases, the checksum value may be in 3 digit hex. When this happens, take only the last two digits only as the checksum value. *For example, if checksum value = 0x2AC, checksum value for the last byte is 0xAC. The "2" will not be used.*

Status Packets

When you send a command that does not cause the DC Load to send requested information back to you, you will receive a status packet back. The structure of a status packet is

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4 to 24	Byte 25
0xAA	Address	0x12	Status byte	Reserved	Checksum

- **0x12** is the status command to indicate that it is a return packet.
- **Status byte** returns the status of the instrument after sending a command. The returned values are defined by the table below:

0x90	Checksum incorrect
0xA0	Parameter incorrect
0xB0	Unrecognized command
0xC0	Invalid command
0x80	Command was successful

Examples

1. Turning on Remote mode to ON

- The command to adjust remote mode is 0x20. The structure of the command is indicated as:

0x20 Set the DC Load to remote operation

Byte offset	Meaning
3	0 means front panel operation. 1 means remote operation.
4-24	Reserved

- To turn on remote mode, the following command needs to be sent to the instrument as follows:

AA 00 20 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CB

- **AA** is the first byte, and **20** indicates the command to send, which is for remote mode. **01** tells the load to turn on remote operation. **CB** is the checksum, meaning **AA + 20 + 1 = CB** in hex.

- If the command is sent correctly, the following string will be in the return packet:
  AA 00 12 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3C
  - **AA** is the first byte, and **12** indicates that it's a return packet. **80** is the status byte, which in this case means command sent was successful. **3C** is the checksum, meaning **AA + 12 + 80 = 3C** in hex.
- If there was an error or the command was not sent correctly, the return string may give you a different value for the **Status byte**. The most common one is **90**, which means the checksum was calculated incorrectly.

2. Setting Maximum Output Voltage

- Some commands seem more complex than they are. Setting maximum voltage is one of them. Below is the command instruction for setting maximum voltage on the load.

0x22 Setting Maximum Output Voltage

Byte offset	**Meaning**
3	Lower low byte of maximum voltage. 1 represents 1 mV.
4	Lower high byte of maximum voltage.
5	Upper low byte of maximum voltage.
6	Upper high byte of maximum voltage.
7-24	Reserved

The description is a bit cryptic, but the example below will explain how to set it up properly.

Example: Set maximum voltage to 16.23 V.

1. According to this command, **1 represents 1 mV**. This is the LSB decimal representation, meaning that each decimal value equates to 1 mV. This will be the conversion formula to follow by. Therefore, in this example 16.23 V would represent **16,230** in decimal (16,230 mV = 16.23 V).
2. Now, this decimal needs to be converted to hexadecimal. **16,230** in hexadecimal equates to **0x00003F6A** in 4 bytes.
3. Since the format is in little-endian format, the 4 bytes of hexadecimal values are arranged as **0x6A3F0000**.
4. According to the above table, Byte offset **3** would be **6A, 4** is **3F**, **5** is **00**, and **6** is **00**.
5. Putting it all together, to set maximum voltage to 16.23 V, the following command is sent:

AA 00 22 6A 3F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 75

- **AA** is always the first byte; **22** is the command byte for setting maximum voltage; **6A, 3F,** and the following 2 bytes of **00**'s represent the 16.23 V value. The remainder of the **00**'s are the reserved bytes and are not used. Hence, the value is just 0. The checksum is **75** since **AA + 22 + 6A + 3F = 175**. Taking the last two values, it becomes **75**.

Additional Notes

- For detailed information on all other supported commands, please refer to user manual.
- If you are programming in a language that supports object oriented programming, a high level python library is available for download at www.bkprecision.com to save time. The library can be used as a COM server object, allowing other common object oriented programming languages to utilize the methods in the object. The library is like an API, and it does all the bit manipulation and data conversion so that setting voltage is as simple as calling a function. It has been tested and successfully used in programming languages such as VB, C#, and of course Python.