Theory of Operation

Below is the Theory of Operation for interfacing with the device using the API defined further below. C# sample code can be made available.

NOTE: Firmware version 3.1.4.7 or later is recommended for the best API response times.

- The device supports two modes of operation, ECHO OFF and ECHO ON. The ECHO OFF mode is the default mode on power up and is required for interacting with the device as indicated below. If the unit was set to ECHO ON mode (echoon) after it was powered up, it must be returned to ECHO OFF mode (echooff) before sending any commands programmatically.
- The device responds to a number of commands (together known as the command-line API) delivered via the USB port (most commonly) or UDP (for WiFi connectivity).
- The USB port is configured on the device as a USB Serial Port. As a result, interfacing can be done with any standard terminal programs (hyperterminal, puTTY, MAC terminal window, etc), or direct serial port programming via C, C#, LabView, MatLab, etc.
    - Serial port settings are:
        - ♣ Baud Rate: 115200
        - ♣ Data Bits: 8
        - ♣ Parity bits = None
        - ♣ Stop bits = 1
        - ♣ Flow Control = None
- The device behaves as follows and in sequence:
    a. Perform analog/digital conversion and mathematical functions as part of optical level detection
        i. Buffer up to 4 characters of any incoming commands while performing the processing above
    b. Check for any characters in the buffer referenced above to indicate an incoming command and process any commands as required. Important programming notes:
        i. It can take up to 50ms for the system to complete complex analog/digital tasks and process the remainder of the command that was not buffered.
        ii. The device determines the completion of the command by detecting a "\r" character, i.e. ASCII code 13 decimal.
    c. Start all over from the top at (a)….
- As a result of the above device behavior, the recommended method to program each device is to:
    - Always append commands with \r to indicate the command completion.
        - ♣ Do NOT send a "\r\n" command as the "\n" will be interpreted as the start of the next command, typically resulting in a delay followed by a "-999\r\n" response indicating an unknown command. Some programming languages have API's that automatically attach a termination character. For example C# SerialPort.WriteLine(…) will append the "\n" by default. In this case use SerialPort.Write(…) with the "\r" embedded in the string passed to the API.
        - ♣ Note that "\r" is not two characters, as would be typed at a terminal or sent as part of a string by some programming API's. It represents the ASCII code 13 or a carriage return.
    - Send the first character of the command, for example the "g" in "getcurrent\r".

        - ♣ It is good practice to “drain” the serial line input prior to sending any commands. This involves simply reading any characters that might be stuck in the serial input buffer prior to sending a new command.
    - Pause 50ms
        - ♣ For firmware versions 3.1.4.7 or greater, this can be relaxed to 10ms.
    - Send the remainder of the command, i.e. “etcurrent\r”.
        - ♣ IMPORTANT: Apple/MAC and some Unix-based systems have shown a need for a 1ms inter-character delay.
    - Immediately start sensing the response from the device.
        - ♣ The device will always send a response to acknowledge the status of the command completion as well as to return values for “get” commands.
            - It is important to always read the response back, even if there is no interest in the response, to empty the serial input buffer.
        - ♣ The response will always be terminated with a “\r\n” (13 decimal, 10 decimal) sequence. This can be used to sense the end of the response and start sending the next command.
        - ♣ Be careful not to expect an immediate response from the device. For instance, after sending the remainder of the command, i.e. “etcurrent\r”, do not immediately check the receive buffer and give-up if data is not sensed. It can take over 1 ms for even the fastest commands to return. It is better (more robust) to (a) send the remainder of the command, (b) continue looking for and processing a response until “\r\n” is sensed, and (c) use a timeout to detect when a response has not returned within some time-out period.
            - Regarding time-outs for command responses, “get” commands will typically respond within 100ms. “set” commands that store their configuration in flash memory can take up to 5 seconds to respond. Other special commands like “setuserdark” and “captureflash” can take longer to respond.
        - ♣ There are certain commands that will return multiple lines, i.e. multiple “\r\n” terminations in response to a single command.
        - ♣ For commands that return a large amount of data, such as getlogdata, ensure that the receiving buffer is large enough to hold all the data.
- Each device is single threaded, meaning that commands and responses need to be processed in sequence. A 2nd command cannot, for example, be initiated before the 1st command’s response is fully processed. As a result, if a multi-threaded application is accessing the device, a programmatic lock must be placed around device command/response sequences to make sure multiple threads do not attempt to access the device at the same time.
    - If multiple devices are being monitored, a single lock can be used for access to all devices. This is simpler from a coding perspective, but it is not as efficient as it does not allow multiple devices to operate in parallel. For the best performance it is recommended that a per-device lock be established. This allows all devices to be accessed in parallel.
    - As a further performance benefit when monitoring multiple devices, the delay after the first character can be performed in parallel across all devices. For example, if sending “getcurrent\r” to 5 devices, one would:
        - ♣ Send “g” to all 5 devices
        - ♣ Wait 50ms
        - ♣ Send “etcurrent\r” to all devices

        - ♣ Process the reply from all devices
            - The replies can be processed in “round-robin” fashion, allowing commands that return a large amount of data, such as getlogdata, to be processed faster.
- Many of the more popular command have had 2-character short-cuts added over time. This allows rapid, repeated access to sensor data because the command fits in the 4-character buffer and does not require a delay after the first character of the command. These commands, and short-cuts, are as follows:
    - gc = getcurrent, introduced in FW version 3.0.5.4
    - gi = getirradiance, introduced in FW version 3.0.5.4
    - gv = getvoltage, introduced in FW version 3.0.5.4
    - gt = gettrans, introduced in FW version 3.0.9.4
    - go = getod, introduced in FW version 3.0.9.4