

**NAME**

amptek – Amptek Pocket MCA 8000/8000A

**INTRODUCTION**

spec communicates with the Amptek MCA-8000 and MCA-8000A pocket MCA devices over an RS-232C interface. The serial communication is non-standard and requires spec to set the DTR and RTS modem-control outputs and read the DSR and CTS modem-control inputs. Thus spec only supports this MCA on platforms with serial ports that respond to the standard modem-control `ioctl()` calls.

spec supports the newer (but still nonstandard) MCA-8000A protocol included on models with serial number 3660 and higher. The new protocol allows the MCA to be used over most USB-to-serial adapters, but currently only on *Linux* platforms. Do not run the older protocol over a USB-to-serial adapter. If it works at all, there will be heavy CPU usage and much slower communications. Note, Amptek can update the firmware of older models. The MCA 8000A currently will not work on a Mac in native mode, although the MCA will work running spec on a *Linux* virtual machine on a Mac if one connects directly to the USB-to-serial adapter through the *Linux* virtual machine. The new protocol requires spec release 5.08.05-6 or higher.

The MCA can be used with baud rates up to 115200, although maximum effective transfer rates seem to be about half that.

The MCA can collect data in up to 16,384 active channels and has memory for 32,768 channels. Memory group sizes can be configured from 256 to 16,384 channels in powers of two, and the number of groups is 32,768 divided by the group size.

The number of usable channels is reduced slightly, though, due to “the sliding scale linearization technique” employed by the MCA ADC. The number of channels in a group is reduced by 1/32 of the group size. For a 256-channel group, the last 8 channels are not useful. For a 16,384-channel group, the last 512 channels are not useful. spec fills those locations with zeroes.

The MCA can be powered by either batteries or an external power source. The power can be turned on remotely by sending a special sequence of signals over the DTR and RTS modem control lines. When spec starts up or on a "reconfig" command, spec sends the signals that will turn on the MCA if had been turned off. When operating on battery, the MCA turns itself off after two minutes of inactivity. While the MCA is acquiring data, spec will normally poll the MCA to check its status, which will keep the MCA from shutting off. If the MCA does shut itself off, spec will send the signals to turn the MCA back on and then retry the command.

## FUNCTIONS

The standard MCA commands supported by `spec` are described in the `mca` help file. Of the standard functions, the Amptek MCA supports `mca_get()` (but not `mca_put()`). The standard `mca_par()` options "info", "chans", "max\_chans", "max\_channels", "disable", "auto\_run", "soft\_preset", "auto\_clear", "native\_type", "preset", "live", "real", "elapsed\_live", "elapsed\_real", "dead", "run", "halt", "clear", "group\_size", "select\_group", "gain" and "chan#", are supported by the Amptek MCA.

The options, "group\_size" and "gain" are synonyms as are "max\_chans" and "max\_channels".

Legal values for the "group\_size" argument are 256, 512, 1024, 2048, 4096, 8192 and 16384. Legal values for the "select\_group" argument are from 0 to  $(32768/\text{group\_size}) - 1$ .

The native data type for the MCA is `ulong`, i.e., unsigned long (32-bit) integers.

The "auto\_clear" mode is on by default. Note, however, it takes time (proportional to the group size) to clear the data. Thus, count times not using a preset will be affected.

Additional `mca_par()` options are as follows:

`mca_par("show_status")` – Reads and displays the current parameters from the MCA status word.

`mca_par("time_stamp")` – Reads the time stamp of the current data group and returns a value in UNIX time format (the number of seconds since the start of Jan 1, 1970). The ASCII representation is available as: `date(mca_par("time_stamp"))`.

`mca_par("time_stamp", value)` – Updates the time stamp to match the time passed in UNIX time format via `value`. If `value` is zero, the current system time is used.

`mca_par("threshold")` – Reads the MCA status and returns the current threshold value.

`mca_par("threshold", value)` – Sets the MCA threshold to `value`. The threshold is the channel number of the lowest channel to be used for data collection. Channels below the threshold channel don't collect data.

`mca_par("battery_type")` – Returns the state of the battery-type flag in the MCA. The battery type is used to calculate the remaining battery power. Battery type zero is 1.5 Volt alkaline cells. Battery type one is

for 1.2 Volt NiCad cells.

`mca_par("battery_type", value)` – Sets the battery-type flag in the MCA. A value of zero selects 1.5 Volt alkaline cells. A nonzero value selects 1.2 Volt NiCad cells. The battery type is only used for choosing which scale to use to calculate the remaining battery capacity.

`mca_par("battery")` – Returns the remaining battery capacity as a percentage based on the battery type and the battery value returned in the MCA status word.

If the magnitude of the counts is known to be less than 65536 in each channel, the read-out time can be cut in half by only reading the two low-order bytes for each data point. `spec` implements a "read\_short" mode to implement the faster read. Note, the native data type remains `ulong`.

`mca_par("read_short")` – Returns nonzero if "read\_short" mode is on, otherwise returns zero.

`mca_par("read_short", value)` – Sets "read\_short" mode to on or off according to whether `value` is nonzero or zero.

#### SEE ALSO

`mca` [www.amptek.com](http://www.amptek.com)