The `pdcaen` package

Maurizio Loreti
Istituto Nazionale di Fisica Nucleare
Sezione di Padova

December 2004

1 Introduction

CAEN[1] builds and sells two power supplies specifically designed for research in High Energy Physics, and that may be remotely controlled by software through a TCP/IP socket connection: the SY1527 and the SY2527. CAEN itself supplies a (C language) library[2] providing software developers with a standard interface for the control of these power supply systems; the current version[3] is 2.11.

2 The `pdcaen` C++ wrapper

In Padova this library has been encapsulated in a C++ interface, the "`pdcaen`" package, here described; that happened because our power supplies had to be driven by a C++ main program, the so-called *long term test* by Wim Beaumont et al., that tests our silicon detectors. The interface itself is declared and defined in the files `sy2527.hh` and `sy2527.cxx`; in addition, other files implement small programs that interacts in various ways with the SY2527 power supply through the `pdcaen` interface.

A `Makefile` is supplied, that drives the C++ compiler in order to build all these programs. All the software conforms with the 1998 C++ standard, ISO/IEC 14882-1998; and has been tested with various versions of both the GNU `g++` compiler (version 3), and of the Kuck & Associates, Inc. (KAI) KCC compiler.

The preprocessor symbols `NDEBUG` and `CAENDEBUG` are tested inside `sy2527.hh`: the first one, if defined, suppresses the explicative error messages sent to the standard error stream if some calls to the `pdcaen` routines controlling the target power supply fail; the second one, if defined, sends to the standard output stream a message summarizing the communication status for *every* call to these routines.

The `pdcaen` package assumes that several power supplies may be addressed: but all of them with the same user name[4] and password; user name and password are static members of the class `caenPST`, declared inside `sy2527.hh`, and must be defined somewhere by the user.

2.1 Utility procedures

The header file `sy2527.hh` defines, in the namespace `sy2527`, two general procedures:

[1] Costruzioni Apparecchiature Elettroniche Nucleari SPA, Viareggio (Italy).
[2] Freely available for download at the URL `http://www.caen.it/computing/scdown.php` .
[3] As of December 2004.
[4] The CAEN defines three user names, "admin", "user" and "guest", with decreasing privileges.

`void version(std::ostream & o = std::cout)`
: Writes on the output stream `o` version informations about the CAEN libraries and the `pcdaen` package; since these version informations cannot be easily obtained from the shared libraries themselves, they are internally defined (as `static const char []`, and inside the namespace `sy2527`), in the file `sy2527.cxx`.

`std::string timeStamp(void)`
: Used for debug printouts; returns a string containing month, day and current time.

2.2 Classes

The header file `sy2527.hh` defines two classes:

`caenPST`
: Abstraction of a CAEN power supply controlled over TCP/IP. The constructor takes two `const std::string &` arguments: a symbolic name used by the CAEN library to identify a particular power supply; and a TCP/IP address. If the costructor can establish a socket connection with the power supply at the given network address, then queries for the available high-voltage boards; if it cannot, an exception of type `std::runtime_error` is thrown. The (virtual) destructor closes the socket connection, and deallocates the resources allocated by the constructor.

  N.B.: copy constructor and assignment operator are declared private and never defined, in order to prevent their use.

`caenBD`
: Abstraction of a CAEN high-voltage board. It is used by `caenPST` internally, and should *not* be addressed by the user directly (but only through the methods of the class `caenPST`).

2.3 Board-related methods

`unsigned int getNBoards(void)`
: Returns the number N of high-voltage boards that the power supply can host (and that will be numbered from 0 to $N-1$).

`double getTemp(unsigned short b)`
: Returns the temperature (in degrees centigrades) of the board `b`.

`unsigned int getBdStatus(unsigned short b)`
: Returns the status of the (existent) board `b` (see the high-voltage board manual for the meaning of every bit).

`caenBD * testB(unsigned int b)`
: Returns `NULL` if there is no high-voltage board in the slot `b`.

`bool testBC(unsigned int b, unsigned short c)`
: Returns `true` if `c` is a valid channel number for the (existent) high-voltage board `b`.

2.4 Channel-related methods: accessors

`double getV(unsigned int b, unsigned short c)`
: Returns the actual voltage of the channel `c` in the (existent) board `b`.

`double getI(unsigned int b, unsigned short c)`
: Returns the actual current of the high-voltage channel `c` in the board `b`.

`double getVx(unsigned int b, unsigned short c)`
`x` may be 0 or 1. Returns the value of `V0` or `V1` for the channel `c` in the board `b`.

`double getIx(unsigned int b, unsigned short c)`
`x` may be 0 or 1. Returns the value of `I0` or `I1` for the channel `c` in the board `b`.

`double getTrip(unsigned int b, unsigned short c)`
Returns the trip time (in seconds) for the channel `c` in the board `b`; see the high-voltage board manual for more.

`double getRup(unsigned int b, unsigned short c)`
Returns the ramp-up voltage gradient (in V/s) for the channel `c` in the board `b`.

`double getRdn(unsigned int b, unsigned short c)`
Returns the ramp-down voltage gradient (in V/s) for the channel `c` in the board `b`.

`double getSVMax(unsigned int b, unsigned short c)`
Returns the maximum allowed high-voltage for the channel `c` in the board `b`.

`unsigned int getChStatus(unsigned int b, unsigned short c)`
Returns the actual status of the high-voltage channel `c` in the board `b` (see the high-voltage board manual for the meaning of every bit).

2.5 Channel-related methods: setters

All these methods return a `bool` value: `true` if the call succeeded, `false` otherwise.

`bool setVx(unsigned int b, unsigned short c, float val)`
`x` may be 0 or 1. Sets the value of `V0` or `V1` for the channel `c` in the board `b`.

`bool setIx(unsigned int b, unsigned short c, float val)`
`x` may be 0 or 1. Sets the value of `I0` or `I1` for the channel `c` in the board `b`.

`bool setTrip(unsigned int b, unsigned short c, float val)`
Sets the value of the trip time (in seconds) for the channel `c` in the board `b`.

`bool setRup(unsigned int b, unsigned short c, float val)`
Sets the ramp-up voltage gradient (in V/s) for the channel `c` in the board `b`.

`bool setRdn(unsigned int b, unsigned short c, float val)`
Sets the value of the ramp-down voltage gradient (in V/s) for the channel `c` in the board `b`.

`bool setSVMax(unsigned int b, unsigned short c, float val)`
Sets the maximum allowed high-voltage for the channel `c` in the board `b`.

`bool setOn(unsigned int b, unsigned short c)`
Sets ON the channel `c` in the board `b`.

`bool setOff(unsigned int b, unsigned short c)`
Sets OFF the channel `c` in the board `b`.

3 The `pdcaen` utility programs

As a general note, all these programs are linked against the CAEN shared libraries; that should be copied somewhere in `LD_LIBRARY_PATH`, and setup with `ldconfig`[5]. For a quick run, you may use the call

```
./doit <prog>
```

that resolves the library symbols of the program `<prog>` against the local copies.

[5] Under Linux; for other operating systems your mileage may vary.

3.1 `tester`

`tester.cxx` is a program that:

1. asks to the user the symbolic name and the network address to be used to construct an instance of the `caenPST` class;
2. asks for a board and a channel number;
3. opens a socket connection with the power supply, sets ON the given channel of the given board, and then drives its voltage to 50 V monitoring the ramp-up every second;
4. after 5 seconds, drives the channel voltage down to 0 V, monitoring the ramp-down every second; then switches it OFF, and exits.

In order to use `tester`:

1. On the *power supply*:
   - The "Power" key must be on **ON, LOCAL**;
   - The "local" switch must be on **LOCAL ENABLE**;
   - The "interlock" switch must be **OFF**.
2. On the *high-voltage board*:
   - A 50 Ω dummy load must be on **HV-ENABLE**;
   - A suitable load (say, 100 MΩ) must be on the selected output channel.

This setup (apart from the dummy load on the channel output) is also required for the other test programs.

3.2 `monitor`

`monitor.cxx` asks for a symbolic name and a network address; then for a board and a channel number. Until `CTRL-C` is pressed, `monitor` will print on the standard output stream the status of the given channel every 5 seconds.

3.3 `caenlogger`

`caenlogger` is a variant of the previous monitoring program: symbolic name, network address, board and channel number are not asked interactively — but defined internally. `caenlogger` requires two command-line options, a file name (may be `/dev/tty`) and a sleep time in seconds: until `CTRL-C` is pressed, at constant time intervals the status of the high-voltage channel is printed in fixed format on the given file.

In Padova, a `php` program reads that output file and builds an active web page where the channel voltage and/or current is shown as a function of the time.

3.4 `caenoff`

This program is called by our UPS in case of power off; it drives all the connected channels of the power supply to 0 V, then switches them OFF.

3.5 `exerciser`

This program asks for a command from the user. Known commands are *Read*, *Set*, *Exit* and *Quit*: the last two commands terminate the program, and the first one reads and prints the status of an high-voltage channel.

With the second command (*Set*), the program asks for what to set: possible answers are *V0*, *I0*, *V1*, *I1*, *Trip*, *Rup*, *Rdn*, *Svmax*, *On*, and *Off*; if the answer was not *On* or *Off*, a numeric value is then required.

For the alphanumeric input, the case is not significant; and only the minimum number of characters needed to identify the command is required. Identifiers and numeric values may be given all on a single line, or in several ones: e.g. `SET <CR> v0 <CR> 50 <CR>` and `s v0 50 <CR>` are equivalent.