Buses in Pyrame

M. Rubio-Roy, F. Magniette

Laboratoire Leprince-Ringuet, École Polytechnique, CNRS/IN2P3
91128 Palaiseau

Pyrame includes two low-level bus modules: cmd_serial and cmd_tcp. The former provides access to USB devices while the latter is an interface for communication via TCP sockets. In addition, a mid-level module: cmd_gpib, provides an interface to PROLOGIX GPIB adapters. Both the USB and Ethernet models are supported via cmd_serial and cmd_tcp.

This documents describes the common API provided by cmd_serial, cmd_tcp and cmd_gpib, as well as the differences between them. The three modules keep a pool of connections.

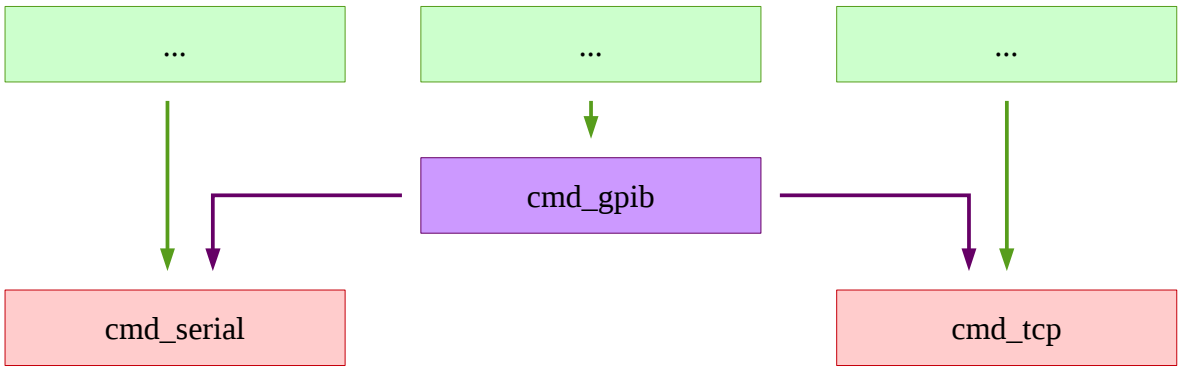

Figure 1: Example of interaction between Pyrame bus modules (gpib, serial, tcp) and others.

Pool and identification

Every link or connection in the pool gets assigned an identification token (id >= 0) which identifies it uniquely during the execution of Pyrame. All functions of gpib, serial and tcp, except init_BUS (e.g.: init_tcp), require this id token as the first argument. The assignment of id is done by this function and constitutes its return value. Along with the id (integer), the modules may also store other information required to access the link. For example, cmd_gpib stores the id of the link returned by cmd_serial or cmd_tcp.

Independently of the exact implementation or programming language, the pool must allow the removal of any arbitrary link without changing the id's of the remaining.

Initialization

The init_MODEL functions take a string (conf_string) as sole parameter to initialize and configure the link. The formats of conf_string for gpib, tcp and serial are:

gpib: slash-separated string of:
    connection_type: usb or eth
    adapter_addr: hostname or IPv4 of GPIB adapter or USB device imprint
    dst_gpib_addr: GPIB address of the destination device
    adapter_gpib_addr (optional; defaults to 0): GPIB address of the adapter
tcp: colon-separated string of:
    host: hostname or IPv4 address
    port: integer representing the TCP port to connect to
serial: USB device imprint

The init_BUS function must perform all the required operations to:

- Initialize a new link either by itself or using one of the bus modules of Pyrame
- Register the new link on its pool
- Return 1 plus an unambiguous id token (on its scope)
- In case of error on any of the steps, deinitialize any possible link or structure already created and return 0 plus an error message

Low-level functions

The low-level bus modules (so far tcp and serial) must include, at least, an implementation of the following functions:

```
init_BUS (conf_string)
deinit_BUS (link_id)
write_BUS (link_id, data)
read_BUS (link_id, bytes_to_read[, timeout])
wrnrd_BUS (link_id, data[, timeout])
expect_BUS (link_id, pattern[, timeout])
```

General considerations:

- Functions return 1/0 for success/failure, plus a string containing either a success/failure message or the result of the operation (e.g. read bytes for read_BUS).
- Functions reading from the BUS (read_BUS, wrnrd_BUS and expect_BUS) must ignore any byte 0 (nul-byte) received.
- Functions reading from the BUS (read_BUS, wrnrd_BUS but not expect_BUS) must read up to a maximum of 4096 bytes (with nul-bytes) and discard bytes 10 and 13.
- Functions writing to the BUS (write_BUS and wrnrd_BUS) must unescape the sequences \e, \t, \n and \r to bytes 27, 9, 10 and 13 respectively.
- Timeout is 60 s by default when the optional parameter is not used. Functions must be able to accept a non-integer timeout.

Expected behavior of the functions in addition to the general considerations:

- write_BUS must send data through the link described by link_id.
- read_BUS must receive bytes_to_read bytes from the link described by link_id in single operation (not byte-per-byte). This read operation must time-out according to the optional parameter or the default time-out value.
- wrnrd_BUS must perform the operations of, first, write_BUS, then, read_BUS(link_id, 4096), on link_id.
- expect_BUS must read byte-per-byte from the link until time-out or pattern is found. Before comparison, pattern is unescaped as on writing functions, resulting in a string of pattern_length bytes. The pattern is found when the last pattern_length bytes received from the link, ignoring nul-bytes, coincides with the unescaped pattern.