

**NAME**

gpib – GPIB (IEEE 488) interface

**BUILT-IN FUNCTIONS**

- `gpib_cntl(a, s)` – Performs the selected GPIB command on the device with address *a*. The string *s* is one of the following:
- "gtl" – Go to local.
  - "llo" – Local lockout.
  - "sdc" – Selected device clear.
  - "dcl" – Device clear (sent to all devices).
  - "get" – Group execute trigger (only sent to the single addressed device).
  - "ifc" – Interface clear. This command resets the GPIB bus by sending the IFC message. The address *a* is ignored. `spec` runs the same code sequence with the "ifc" command as it does when it initializes the GPIB controller on start up or on the `reconfig` command. For most controllers, `spec` sleeps for some fraction of a second after resetting the bus. Also, for most controllers, `spec` asserts the REN (remote enable) command after sending IFC.
- `gpib_get(a)` – Returns a string from the GPIB device with address *a*. Requires the device to terminate the string with a newline, although the trailing newline (and carriage return, if present) is removed from the string before it is returned.
- `gpib_get(a, n)` – As above, but reads *n* bytes and doesn't look for a terminator.
- `gpib_get(a, eos)` – As above, but reads up until a terminator given by the first character of the string *eos*, except for the special cases described below. The terminator is removed.
- `gpib_get(a, d)` – Reads incoming bytes into the data array *d*. The size of *d* determines how many bytes are to be read. Sub-array syntax can be used to limit the number of bytes. The function returns the number of array elements read, or zero if the read times out. Note, no byte re-ordering is done for short- or-long integer data, and no format conversions are done for float or double data.
- `gpib_get(a, mode)` – If *mode* is the string "byte", reads and returns one unsigned binary byte. If *mode* is the string "short", reads two binary bytes and returns the short integer so formed. If *mode* is the string "long", reads four binary bytes and returns the long integer so formed. The last two modes work the same on both *big-endian* and *little-endian* platforms. On both, the incoming data is treated as *big endian*. If the incoming data is *little endian*, use "short\_swap" or "long\_swap". (For `spec` versions prior to release 5.01.01, use `int2` for short and `int4` for long.)
- `gpib_poll(a)` – Returns the serial-poll status from the device with address *a*.
- `gpib_put(a, s)` – Writes the string *s* to the GPIB device with address *a*. Returns the number of bytes written.
- `gpib_put(a, d, [cnt])` – Writes the contents of the data array *d* to the GPIB device with address *a*. By default, the entire array (or subarray, if specified) will be sent. The optional third argument *cnt* can be used to specify the number of array elements to send. For short and long integer arrays, the data will be sent using native byte order. The "swap" option of the `array_op()` function can be used to change the byte order, if necessary. No format conversions are available for float or double data. Returns the number of bytes written.
- `gpib_put(a, "")` – Performs a listener check at device address *a*. Writing a null string (zero-length message) generates a return value of 1 if there is a listener, zero if there is no listener and -1 if the GPIB controller doesn't support the test (as of `spec` release

5.07.03-4).

`gpib_par(addr, "timeout" [, t])` – Returns or sets the timeout for the device with address `addr`. The units are seconds. A value less than or equal to zero resets the timeout to the default value of three seconds. Timeouts are per device (rather than per controller). The timeouts only apply to the user-level GPIB commands. Timeouts used by built-in hardware support will continue to apply to that access. (As of spec release 5.09.02-3.)

### MULTIPLE GPIB CONTROLLERS

spec allows up to eight GPIB controllers to be configured by the same instance of spec. In the configuration editor, use the `^F` and `^B` command on the GPIB controller line of the Interfaces screen to configure each controller. To include a unit number in a GPIB address for a device, enter the address using the notation `unit:address`. An absent `unit:` prefix implies GPIB controller unit zero.

In the `config` file, the unit number associated with a GPIB controller is specified with `@gpib_0`, `@gpib_1`, etc. Also, GPIB unit numbers are encoded using

```
unit × 100 + address.
```

for the GPIB address. From spec, the `gpib_get()`, `gpib_put()`, etc., commands described above, can use either the above coding, the syntax `"unit:address"` (the quotes are required) or the syntax `unit.address` (as of release 5.07.02-2).

### SECONDARY GPIB ADDRESS

Secondary GPIB addresses are supported for all the above functions. The address argument must be passed as a string however, of the form `"pri_sec"`, where `pri` is the primary GPIB address and `sec` is the secondary GPIB address. For example,

```
print gpib_get("3_12")
```

will display the string read from the device with a primary GPIB address of 3 and a secondary GPIB address of 12. There is no provision to include secondary addresses as part of the device configuration information contained in the hardware `config` file.

### SHARING GPIB CONTROLLERS

spec communicates with the GPIB controllers at board (rather than device) level. That makes it unlikely that an arbitrary program could also use the same GPIB controller successfully simultaneously. However, spec can share a GPIB controller among multiple instances of itself. The state of the controller is communicated between the processes using the inter-process communications (IPC) semaphore and shared-memory facilities. Thus sharing is only available on platforms that include these facilities in the kernel.

When sharing multiple GPIB controllers, please note it is essential each shared controller have the same GPIB controller unit number in the spec configuration file.

Note that spec uses only one semaphore to regulate access to all the shared controllers. Thus, if there are two GPIB controllers configured as shared, when any process accesses either controller, other processes will be blocked from accessing both controllers until the first process releases the semaphore. Note also that when the Kinetic Systems KS-3929 SCSI-to-CAMAC controller is configured as a shared controller, the same semaphore is again used.

### SEE ALSO

`serial config_adm`