Linux-GPIB 4.3.4 Documentation

Frank Mori Hess

fmhess@users.sourceforge.net

Dave Penkler

dpenkler@gmail.com

1. Copying

2. Configuration

Configuration of the GPIB library is accomplished through the configuration file gpib.conf, and the administration program gpib_config.

gpib.conf

Name

`gpib.conf` — GPIB library configuration file

Description

The library, and the administration tool gpib_config read their configuration information from the file gpib.conf. By default, file is located in the sysconfdir directory configured when linux-gpib was compiled. The sysconfdir is typically set to /etc or /usr/local/etc. A template gpib.conf file can be found in the util/templates/ subdirectory of the linux-gpib package.

The default config file may be overriden by setting the IB_CONFIG environment variable to a custom file path.

The configuration file must contain one ’interface’ entry for each of the board minors that are going to be used, and can contain zero or more ’device’ entries. ’device’ entries are only required if you wish to open device descriptors with ibfind() instead of using ibdev(). Several example entries, and a table summarizing the possible options follow.

```
interface {
 minor = 0
 board_type = "ni_pci"
 pad = 0
 master = yes
}

interface {
 minor = 1
 board_type = "ines_pci"
 name = "joe"
 pad = 5
 sad = 0
 timeout = T10s
 pci_bus = 0
 pci_slot = 0xd
 master = no
}

interface {
 minor = 2
 board_type = "pcII"
 pad = 3
 sad = 0x62
 eos = 0x0d
 set-reos = yes
 set-bin = no
 set-xeos = no
 set-eot = yes
 base = 0x300
 irq  = 5
 dma  = 0
 master = no
```

```
}

device {
 minor = 0
 name = "counter"
 pad = 24
}

device {
 minor = 0
 name = "voltmeter"
 pad = 7
 sad = 110
 eos = 0xa
 set-reos = yes
 set-bin = no
 set-xeos = yes
 set-eot = no
 timeout = T1s
}
```

Table 1. configuration options

gpib_config

Name

`gpib_config` — GPIB administration program

Synopsis

gpib_config [--minor *number*]

gpib_config [--board-type *board_type*] [--device-file *file_path*] [--dma *number*] [--file *file_path*] [--help] [--iobase *number*] [--ifc] [--no-ifc] [--init-data *file_path*] [--irq *number*] [--minor *number*] [--offline] [--pad *number*] [--pci-bus *number*] [--pci-slot *number*] [--sad *number*] [--serial-number *serial_number*] [--sre] [--no-sre] [--sysfs-device-path *sysfs_device_path*] [--system-controller] [--no-system-controller] [--version]

Description

gpib_config must be run after the kernel driver module for a GPIB interface board is loaded. It performs configuration of driver settings that cannot be performed by libgpib at runtime. This includes configuration which requires root privilege (for example, setting the base address or irq of a board), and configuration which should only be performed once and not automatically redone every time a program using libgpib is run (for example, setting the board's GPIB address).

The board to be configured by gpib_config is selected by the `--minor` option. By default, the board settings are read from the gpib.conf configuration file. However, individual settings can be overiden by use of command-line options (see below).

Options

`-t, --board-type` *board_type*

Set board type to *board_type*.

`-c, --device-file` *file_path*

Specify character device file path for the board. This can be used as an alternative to the --minor option.

`-d, --dma` *number*

Specify isa dma channel *number* for boards without plug-and-play cabability.

`-f, --file` *file_path*

Specify file path for configuration file. The values in the configuration file will be used as defaults for unspecified options. If this option is not specified, then the value of the IB_CONFIG environment variable will be used if it is set. Otherwise, the default configuration file is "sysconfdir/gpib.conf" where sysconfdir is specified when the library is configured before compilation (it is usually /etc or /usr/local/etc).

`-h, --help`

Print help on options and exit.

`-I, --init-data` *`file_path`*

Upload binary initialization data (firmware) from *`file_path`* to board.

`--[no-]ifc`

Perform (or not) interface clear after bringing board online. Default is `--ifc`.

`-b, --iobase` *`number`*

Set io base address to *`number`* for boards without plug-and-play cabability.

`-i, --irq` *`number`*

Specify irq line *`number`* for boards without plug-and-play cabability.

`-m, --minor` *`number`*

Configure gpib device file with minor number *`number`* (default is 0).

`-o, --offline`

Unconfigure an already configured board, don't bring board online.

`-p, --pad` *`number`*

Specify primary gpib address. *`number`* should be in the range 0 through 30.

`-u, --pci-bus` *`number`*

Specify pci bus *`number`* to select a specific pci board. If used, you must also specify the pci slot with `--pci-slot`.

`-l, --pci-slot` *`number`*

Specify pci slot *`number`* to select a specific pci board. If used, you must also specify the pci bus with `--pci-bus`.

`-s, --sad` *`number`*

Specify secondary gpib address. *`number`* should be 0 (disabled) or in the range 96 through 126 (0x60 through 0x7e hexadecimal).

`--[no-]sre`

Assert (or not) remote enable line after bringing board online. Default is `--sre`.

`-a, --sysfs-device-path` *`dev_path`*

Select a specific board to attach by its sysfs device path. The sysfs device path is the absolute path to the device's directory under /sys/devices, with the leading "/sys" stripped off. The device path is available in udev scripts as the DEVPATH variable.

`--[no-]system-controller`

Configure board as system controller (or not).

`-v, --version`

Prints the current linux-gpib version and exits.

3. Supported Hardware

3.1. Supported Hardware Matrix

Table 2. Linux-GPIB Supported Hardware Matrix

3.2. Board-Specific Notes

3.2.1. Agilent (HP) 82341

After power-up, the Agilent 82341 boards require a firmware upload before they can be used. This can be accomplished using the "--init-data" option of gpib_config. The firmware data for the boards can be found at this repository (https://github.com/fmhess/linux_gpib_firmware). Note the C and D versions use different firmware data.

If you specify a non-zero base address in gpib.conf, the driver will assume you are trying to configure a 82341C. Otherwise, the driver will use the kernel's ISAPNP support to attempt to configure an 82341D.

The 82341 does not support detection of an end-of-string character in hardware, it only automatically detects the when the EOI line is asserted. Thus if you use the REOS flag for a read, the board's fifos will not be used for the transfer. This will greatly reduce the maximum transfer rate for your board (which may or may not be noticeable depending on the device you are talking to).

3.2.2. Agilent 82350A/B and 82351A

The Agilent 82350A/B and 82351A boards do not support detection of an end-of-string character during reads in hardware, they can only detect assertion of the EOI line. Thus if you use the REOS flag for a read, the boards' fifos will not be used for the transfer. This will greatly reduce the maximum transfer rate for your board (which may or may not be noticeable depending on the device you are talking to).

After power-up, the 82350A boards require a firmware upload before they can be used. This can be accomplished using the "--init-data" option of gpib_config. The firmware data for the 82350A can be found at this repository (https://github.com/fmhess/linux_gpib_firmware). The 82350B and 82351A do not require a firmware upload.

3.2.3. Agilent 82357A/B

The Agilent 82357A and 82357B require a firmware upload (before gpib_config is run) to become functional after being plugged in. The linux-gpib tarball contains udev rules for automatically running the fxload program to upload the firmware (and to run gpib_config after the firmware is uploaded). However, the actual firmware data itself must be obtained seperately. It can be found at this repository (https://github.com/fmhess/linux_gpib_firmware).

The 82357A/B have a few limitation due to their firmware code:

- They cannot be run as a device, but must be the system controller.
- They cannot be assigned a secondary address.
- They cannot do 7 bit compares when looking for an end-of-string character (they always compare all 8 bits).

3.2.4. Beiming F/S82357

Linux-gpib support requires a minimum firmware version of 1.10 for the F82357 and version 1.20 for the S82357. These devices have on-board firmware and do not require a firmware upload before becoming functional afer plug-in. The on-board firmware can be re-flashed; contact the manufacturer for firmware and re-flash procedure.

Limitations:

- These devices can only be used as system controllers.
- They can only do 8-bit end-of-string (EOS) compares.

3.2.5. fmh_gpib_core

fmh_gpib_core is a GPIB chip written in VHDL suitable for programming into a FPGA. The code for the chip may be found at https://github.com/fmhess/fmh_gpib_core. It supports a cb7210.2 style register interface with some extensions. More specifically, the driver is for the hardware layout specified in src/example/fmh_gpib_top.vhd file in the fmh_gpib_core repository.

The driver obtains its hardware information (base addresses, interrupt, dma, etc.) from the device tree. It expects to find two i/o memory resources, an interrupt, and a dma channel. One i/o memory resource is called "gpib_control_status" which contains the 8 bit cb7210.2 registers. The other i/o memory resource is called "dma_fifos" and contains 16 bit registers for the fifos and transfer counter. The dma channel the chip is wired to is specified with the standard "dmas" and "dma-names" fields, with a dma-name of "rxtx". So, the device tree entry for a chip connected to channel 2 of dma controller "dmac" might look something like:

```
fmh_gpib_0: fmh_gpib@0x00049800 {
                compatible = "fmhess,fmh_gpib_core";
                reg = < 0x00049600 0x00000080
                        0x00049800 0x00000008 >;
                reg-names = "gpib_control_status", "dma_fifos";
                interrupt-parent = < &intc >;
                interrupts = < 0 57 4 >;
                dmas = < &dmac 2 >;
                dma-names = "rxtx";
            }; //end fmh_gpib@0x00049800 (fmh_gpib_0)
```

3.2.6. Self-made usb-gpib adapter

This usb-gpib adapter can be assembled following the project from the Laboratory of Photovoltaics and Optoelectronics at the Faculty of Electrical Engineering, University of Ljubljana. It is available at http://lpvo.fe.uni-lj.si/gpib . The adapter allows the control of GPIB devices with some limitations: it can only be the system controller; multicontroller and device operations are not supported (as yet). The linux-gpib driver 'lpvo_usb_gpib', written at the Department of Physics of University of Florence (Italy), is currently under development. It offers basic capabilities like ibrd(), ibwrt(), WaitSRQ() and others. Requests for unsupported features are flagged by a diagnostic message to syslog.

3.2.7. GPIO bitbang adapter

WARNING: this driver is incomplete and its behavior does not conform to the IEEE 488 standards.

This is a simple GPIO bitbang driver that currently only works on Raspberry Pi platforms. For information on the GPIO to GPIB pinouts as well as limitations see the source code at:

```
linux-gpib-kernel/drivers/gpib/gpio/gpib_bitbang.c
```

Two pin maps between gpio and gpib are currently supported: elektronomikon and gpib4pi-1.1. For schematics, boards and information on driver IC's see http://elektronomikon.org and lightside-instruments gpib4pi (https://www.hackster.io/lightside-instruments/the-gpib4pi-gpib-for-raspberry-pi-hat-4b3e9a)

The pin_maps are configured via the pin_map module parameter. By default support for the elekronomikon pin_map with SN75160/161 driver IC's is enabled. i.e. the default module parameter settings are pin_map=elektonomikon and sn7516x_used=1. To disable support for the IC's when they are not installed the module parameter sn7516x_used must be set to zero:

```
modprobe gpib_bitbang sn7516x_used=0
```

To set the pin_map for the Lightside instruments gpib4pi-1.1 board

```
modprobe gpib_bitbang pin_map="gpib4pi-1.1"
```

3.2.8. National Instruments GPIB-USB-B

The USB-B requires a firmware upload (before gpib_config is run) to become functional after being plugged in. The linux-gpib tarball contains udev rules for automatically running the fxload program to upload the firmware (and to run gpib_config after the firmware is uploaded). However, the actual firmware must be obtained separately. It can be found at this repository (https://github.com/fmhess/linux_gpib_firmware).

3.2.9. National Instruments GPIB-USB-HS and GPIP-USB-HS+

Unlike the USB-B, the USB-HS does not require a firmware upload to become functional after being plugged in. Most GPIB-USB-HS+ also do not require firmware upload, however some exceptions have been identified. If your GPIB-USB-HS+ initially comes up with a USB product id of 0x761e it will require a one-time firmware upload which permanently changes the product id to the usual 0x7618 for a GPIB-USB-HS+. Currently this can be done by plugging the adapter into a Windows computer which has the NI driver software installed. Alternatively, you may use the hsplus_load (https://github.com/fmhess/hsplus_load) utility to initialize the adapter under Linux.

The linux-gpib tarball contains udev rules which will automatically run gpib_config after the device is plugged in.

Beware of inexpensive NI GPIB-USB-HS clones. While they work on Windows most will not and cannot be made to work with linux-gpib. See the end of the discussion on bug ticket #82 (https://sourceforge.net/p/linux-gpib/bugs/82/).

4. Linux-GPIB Reference

Reference for libgpib functions, macros, and constants.

4.1. Global Variables

ibcnt and ibcntl

Name

`ibcnt and ibcntl` — hold number of bytes transferred, or errno

Synopsis

```
#include <gpib/ib.h>

volatile int ibcnt;
volatile long ibcntl;
```

Description

ibcnt and ibcntl are set after IO operations to the the the number of bytes sent or received. They are also set to the value of errno after EDVR or EFSO errors.

If you wish to avoid using a global variable, you may instead use ThreadIbcnt() or ThreadIbcntl() which return thread-specific values.

iberr

Name

`iberr` — holds error code

Synopsis

```
#include <gpib/ib.h>

volatile int iberr;
```

Description

iberr is set whenever a function from the 'traditional' or 'multidevice' API fails with an error. The meaning of each possible value of iberr is summarized in the following table:

Table 1. iberr error codes

If you wish to avoid using a global variable, you may instead use ThreadIberr() which returns a thread-specific value.

ibsta

Name

`ibsta` — holds status

Synopsis

```
#include <gpib/ib.h>

volatile int ibsta;
```

Description

ibsta is set whenever a function from the 'traditional' or 'multidevice' API is called. Each of the bits in ibsta has a different meaning, summarized in the following table:

Table 1. ibsta Bits

If you wish to avoid using a global variable, you may instead use ThreadIbsta() which returns a thread-specific value.

4.2. 'Traditional' API Functions

ibask

Name

`ibask` — query configuration (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibask(int ud, int option, int *result);
```

Description

Queries various configuration settings associated with the board or device descriptor *ud*. The *option* argument specifies the particular setting you wish to query. The result of the query is written to the location specified by *result*. To change the descriptor's configuration, see ibconfig().

Table 1. ibask options

Return value

The value of ibsta is returned.

ibbna

Name

`ibbna` — change access board (device)

Synopsis

```
#include <gpib/ib.h>
int ibbna(int ud, const char *name);
```

Description

ibbna() changes the GPIB interface board used to access the device specified by *ud*. Subsequent device level calls using the descriptor *ud* will assume the device is connected to the interface board specified by *name*. If you wish to specify a device's new access board by board index instead of name, you can use the IbcBNA option of ibconfig().

The name of a board can be specified in the configuration file gpib.conf.

On success, iberr is set to the board index of the device's old access board.

Return value

The value of ibsta is returned.

ibcac

Name

ibcac — assert ATN (board)

Synopsis

```
#include <gpib/ib.h>
int ibcac(int ud, int synchronous);
```

Description

ibcac() causes the board specified by the board descriptor *ud* to become active controller by asserting the ATN line. The board must be controller-in-change in order to assert ATN. If *synchronous* is nonzero, then the board will wait for a data byte on the bus to complete its transfer before asserting ATN. If the synchronous attempt times out, or *synchronous* is zero, then ATN will be asserted immediately.

It is generally not necessary to call ibcac(). It is provided for advanced users who want direct, low-level access to the GPIB bus.

Return value

The value of ibsta is returned.

ibclr

Name

ibclr — clear device (device)

Synopsis

```
#include <gpib/ib.h>
int ibclr(int ud);
```

Description

ibclr() sends the clear command to the device specified by *ud*.

Return value

The value of ibsta is returned.

ibcmd

Name

`ibcmd` — write command bytes (board)

Synopsis

```
#include <gpib/ib.h>
int ibcmd(int ud, const void *commands, long num_bytes);
```

Description

ibcmd() writes the command bytes contained in the array *commands* to the bus. The number of bytes written from the array is specified by *num_bytes*. The *ud* argument is a board descriptor, and the board must be controller-in-charge. Most of the possible command bytes are declared as constants in the header files. In particular, the constants GTL, SDC, PPConfig, GET, TCT, LLO, DCL, PPU, SPE, SPD, UNL, UNT,and PPD are available. Additionally, the inline functions MTA(), MLA(), MSA(), and PPE_byte() are available for producing 'my talk address', 'my listen address', 'my secondary address', and 'parallel poll enable' command bytes respectively.

It is generally not necessary to call ibcmd(). It is provided for advanced users who want direct, low-level access to the GPIB bus.

Return value

The value of ibsta is returned.

ibcmda

Name

`ibcmda` — write command bytes asynchronously (board)

Synopsis

```
#include <gpib/ib.h>
int ibcmda(int ud, const void *commands, long num_bytes);
```

Description

ibcmda() is similar to ibcmd() except it operates asynchronously. ibcmda() does not wait for the sending of the command bytes to complete, but rather returns immediately.

While an asynchronous operation is in progress, most library functions will fail with an EOIP error. In order to sucessfully complete an asynchronous operation, you must call ibwait() with CMPL set in the wait mask, until the CMPL bit is set ibsta. Asynchronous operations may also be aborted with an ibstop() or ibonl() call.

After the asynchronous I/O has completed and the results resynchronized with the current thread, the Linux-GPIB extensions AsyncIbsta, AsyncIberr, AsyncIbcnt and AsyncIbcntl may be useful to more cleanly separate the results of the asynchronous I/O from the results of the ibwait or similar call used to resynchronize.

Return value

The value of ibsta is returned.

ibconfig

Name

`ibconfig` — change configuration (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibconfig(int ud, int option, int setting);
```

Description

Changes various configuration settings associated with the board or device descriptor `ud`. The `option` argument specifies the particular setting you wish to modify. The `setting` argument specifies the option's new configuration. To query the descriptor's configuration, see ibask().

Table 1. ibconfig options

Return value

The value of ibsta is returned.

ibdev

Name

ibdev — open a device (device)

Synopsis

```
#include <gpib/ib.h>
int ibdev(int board_index, int pad, int sad, int timeout, int send_eoi, int
eos);
```

Description

ibdev() is used to obtain a device descriptor, which can then be used by other functions in the library. The argument *board_index* specifies which GPIB interface board the device is connected to. The *pad* and *sad* arguments specify the GPIB address of the device to be opened (see ibpad() and ibsad()). The timeout for io operations is specified by *timeout* (see ibtmo()). If *send_eoi* is nonzero, then the EOI line will be asserted with the last byte sent during writes (see ibeot()). Finally, the *eos* argument specifies the end-of-string character and whether or not its reception should terminate reads (see ibeos()).

Return value

If sucessful, returns a (non-negative) device descriptor. On failure, -1 is returned.

ibeos

Name

ibeos — set end-of-string mode (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibeos(int ud, int eosmode);
```

Description

ibeos() is used to set the end-of-string character and mode. The least significant 8 bits of *eosmode* specify the eos character. You may also bitwise-or one or more of the following bits to set the eos mode:

Table 1. End-of-String Mode Bits

Return value

The value of ibsta is returned.

ibeot

Name

`ibeot` — assert EOI with last data byte (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibeot(int ud, int send_eoi);
```

Description

If *send_eoi* is non-zero, then the EOI line will be asserted with the last byte sent by calls to ibwrt() and related functions.

Return value

The value of ibsta is returned.

ibevent

Name

`ibevent` — get events from event queue (board)

Synopsis

```
#include <gpib/ib.h>
int ibevent(int ud, short *event);
```

Description

ibevent() is used to obtain the oldest event stored in the event queue of the board specified by the board descriptor *`ud`*. The EVENT bit of ibsta indicates that the event queue contains 1 or more events. An event may be a clear command, a trigger command, or reception of an interface clear. The type of event is stored in the location specified by *`event`* and may be set to any of the following values:

Table 1. events

The event queue is disabled by default. It may be enabled by a call to ibconfig(). Each interface board has a single event queue which is shared across all processes and threads. So, only one process can retrieve any given event from the queue. Also, the queue is of finite size so events may be lost (ibevent() will return an error) if it is neglected too long.

Return value

The value of ibsta is returned.

ibfind

Name

`ibfind` — open a board or device (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibfind(const char *name);
```

Description

ibfind() returns a board or device descriptor based on the information found in the configuration file. To suppress error messages printed to stderr by ibfind() set the evironment variable IB_NO_ERROR before calling the routine. It is not required to use ibfind(), since device descriptors can be obtained with ibdev() and the 'board index' (minor number in the configuration file) can be used directly as a board descriptor.

Return value

If sucessful, returns a (non-negative) board or device descriptor. On failure, -1 is returned.

ibgts

Name

`ibgts` — release ATN (board)

Synopsis

```
#include <gpib/ib.h>
int ibgts(int ud, int shadow_handshake);
```

Description

ibgts() is the complement of ibcac(), and causes the board specified by the board descriptor *ud* to go to standby by releasing the ATN line. The board must be controller-in-change to change the state of the ATN line. If *shadow_handshake* is nonzero, then the board will handshake any data bytes it receives until it encounters an EOI or end-of-string character, or the ATN line is asserted again. The received data is discarded.

It is generally not necessary to call ibgts(). It is provided for advanced users who want direct, low-level access to the GPIB bus.

Return value

The value of ibsta is returned.

ibist

Name

`ibist` — set individual status bit (board)

Synopsis

```
#include <gpib/ib.h>
int ibist(int ud, int ist);
```

Description

If *ist* is nonzero, then the individual status bit of the board specified by the board descriptor *ud* is set. If *ist* is zero then the individual status bit is cleared. The individual status bit is sent by the board in response to parallel polls.

On success, iberr is set to the previous ist value.

Return value

The value of ibsta is returned.

iblines

Name

`iblines` — monitor bus lines (board)

Synopsis

```
#include <gpib/ib.h>
int iblines(int ud, short *line_status);
```

Description

iblines() is used to obtain the status of the control and handshaking bus lines of the bus. The board used to monitor the bus is specified by the *ud* argument, and the status of the various bus lines are written to the location specified by *line_status*.

Some older chips are not capable of reporting the status of the bus lines, so each line has two corresponding bits in *line_status*. One bit indicates if the board can monitor the line, and the other bit indicates the line's state. The meaning of the *line_status* bits are as follows:

Table 1. line status bits

Return value

The value of ibsta is returned.

ibln

Name

`ibln` — check if listener is present (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibln(int ud, int pad, int sad, short *found_listener);
```

Description

ibln() checks for the presence of a device, by attempting to address it as a listener. *ud* specifies the GPIB interface board which should check for listeners. If *ud* is a device descriptor, then the device's access board is used.

The GPIB address to check is specified by the *pad* and *sad* arguments. *pad* specifies the primary address, 0 through 30 are valid values. *sad* gives the secondary address, and may be a value from 0x60 through 0x7e (96 through 126), or one of the constants NO_SAD or ALL_SAD. NO_SAD indicates that no secondary addressing is to be used, and ALL_SAD indicates that all secondary addresses should be checked.

If the board finds a listener at the specified GPIB address(es), then the variable specified by the pointer *found_listener* is set to a nonzero value. If no listener is found, the variable is set to zero.

The board must be controller-in-charge to perform this function. Also, it must have the capability to monitor the NDAC bus line (see iblines()).

This function has the additional effect of addressing the board as talker for the duration of the Find Listeners protocol, which is beyond what IEEE 488.2 specifies. This is done because some boards cannot reliably read the state of the NDAC bus line unless they are the talker. Being the talker causes the board's gpib transceiver to configure NDAC as an input, so its state can be reliably read from the bus through the transceiver.

Return value

The value of ibsta is returned.

ibloc

Name

`ibloc` — go to local mode (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibloc(int ud);
```

Description

Causes the board or device specified by the descriptor *ud* to go to local mode. If *ud* is a board descriptor, and the board is in local lockout, then the function will fail.

Note, if the system controller is asserting the REN line, then devices on the bus will return to remote mode the next time they are addressed by the controller in charge.

Return value

The value of ibsta is returned.

ibonl

Name

`ibonl` — close or reinitialize descriptor (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibonl(int ud, int online);
```

Description

If the *online* is zero, then ibonl() frees the resources associated with the board or device descriptor *ud*. The descriptor cannot be used again after the ibonl() call.

If the *online* is nonzero, then all the settings associated with the descriptor (GPIB address, end-of-string mode, timeout, etc.) are reset to their 'default' values. The 'default' values are the settings the descriptor had when it was first obtained with ibdev() or ibfind().

Return value

The value of ibsta is returned.

ibpad

Name

ibpad — set primary GPIB address (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibpad(int ud, int pad);
```

Description

ibpad() sets the GPIB primary address to *pad* for the device or board specified by the descriptor *ud*. If *ud* is a device descriptor, then the setting is local to the descriptor (it does not affect the behaviour of calls using other descriptors, even if they refer to the same physical device). If *ud* is a board descriptor, then the board's primary address is changed immediately, which is a global change affecting anything (even other processes) using the board. Valid GPIB primary addresses are in the range from 0 to 30.

Return value

The value of ibsta is returned.

ibpct

Name

`ibpct` — pass control (board)

Synopsis

```
#include <gpib/ib.h>
int ibpct(int ud);
```

Description

ibpct() passes control to the device specified by the device descriptor *ud*. The device becomes the new controller-in-charge.

Return value

The value of ibsta is returned.

ibppc

Name

`ibppc` — parallel poll configure (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibppc(int ud, int configuration);
```

Description

Configures the parallel poll response of the device or board specified by *ud*. The *configuration* should either be set to the 'PPD' constant to disable parallel poll responses, or set to the return value of the PPE_byte() inline function to enable and configure the parallel poll response.

If *ud* is a device descriptor then the device will be remotely configured by the controller.

If *ud* is a board descriptor then the board will be locally configured. Note, in order to do a local parallel poll configuration IbcPP2 must be set using ibconfig(). IEEE 488.2 prohibits local parallel poll configuration (IEEE 488.1 PP2 subset), requiring support for remote parallel poll configuration (IEEE 488.1 PP1 subset) instead.

After configuring the parallel poll response of devices on a bus, you may use ibrpp() to parallel poll the devices.

Return value

The value of ibsta is returned.

ibrd

Name

`ibrd` — read data bytes (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibrd(int ud, void *buffer, long num_bytes);
```

Description

ibrd() is used to read data bytes from a device or board. The argument *ud* can be either a device or board descriptor. Up to *num_bytes* bytes are read into the user-supplied array *buffer*. The read may be terminated by a timeout occuring(see ibtmo()), the talker asserting the EOI line, the board receiving the end-of-string character (see ibeos()), receiving a device clear command, or receiving an interface clear.

If `ud` is a device descriptor, then the library automatically handles addressing the device as talker and the interface board as listener before performing the read.

If `ud` is a board descriptor, no addressing is performed and the board must be addressed as a listener by the controller-in-charge.

After the ibrd() call, ibcnt and ibcntl are set to the number of bytes read.

Return value

The value of ibsta is returned.

ibrda

Name

`ibrda` — read data bytes asynchronously (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibrda(int ud, void *buffer, long num_bytes);
```

Description

ibrda() is similar to ibrd() except it operates asynchronously. ibrda() does not wait for the reception of the data bytes to complete, but rather returns immediately.

While an asynchronous operation is in progress, most library functions will fail with an EOIP error. In order to sucessfully complete an asynchronous operation and resynchronize its results with the current thread, you must call ibwait() with CMPL set in the wait mask, until the CMPL bit is set ibsta. Asynchronous operations may also be completed by a call to ibstop() or ibonl() call. Note, ibwait() will only complete the asynchronous operation if you explicitly set the CMPL bit in the wait mask parameter of ibwait().

After the asynchronous I/O has completed and the results resynchronized with the current thread, the Linux-GPIB extensions AsyncIbsta, AsyncIberr, AsyncIbcnt and AsyncIbcntl may be useful to more cleanly separate the results of the asynchronous I/O from the results of the ibwait or similar call used to resynchronize.

Return value

The value of ibsta is returned.

ibrdf

Name

`ibrdf` — read data bytes to file (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibrdf(int ud, const char *file_path);
```

Description

ibrdf() is similar to ibrd() except that the data bytes read are stored in a file instead of an array in memory. *`file_path`* specifies the save file. If the file already exists, the data will be appended onto the end of the file.

Return value

The value of ibsta is returned.

ibrpp

Name

`ibrpp` — perform a parallel poll (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibrpp(int ud, char *ppoll_result);
```

Description

ibrpp() causes the interface board to perform a parallel poll, and stores the resulting parallel poll byte in the location specified by `ppoll_result`. Bits 0 to 7 of the parallel poll byte correspond to the dio lines 1 to 8, with a 1 indicating the corresponding dio line is asserted. The devices on the bus you wish to poll should be configured beforehand with ibppc(). The board which performs the parallel poll must be controller-in-charge, and is specified by the descriptor `ud`. If `ud` is a device descriptor instead of a board descriptor, the device's access board performs the parallel poll.

Return value

The value of ibsta is returned.

ibrsc

Name

`ibrsc` — request system control (board)

Synopsis

```
#include <gpib/ib.h>
int ibrsc(int ud, int request_control);
```

Description

If `request_control` is nonzero, then the board specified by the board descriptor `ud` is made system controller. If `request_control` is zero, then the board releases system control.

The system controller has the ability to assert the REN and IFC lines, and is typically also the controller-in-charge. A GPIB bus may not have more than one system controller.

Return value

The value of ibsta is returned.

ibrsp

Name

`ibrsp` — read status byte / serial poll (device)

Synopsis

```
#include <gpib/ib.h>
int ibrsp(int ud, char *result);
```

Description

ibrsp() obtains the status byte from the device specified by *`ud`*. The status byte is stored in the location specified by *`result`*.

If automatic serial polling is enabled on the board controlling the device, the status byte is automatically read and queued whenever the device requests service. If the status byte queue is not empty `ibrsp()` obtains the status byte information from the queue. If the queue is empty the status byte is obtained by serial polling the device. Automatic serial polling is controlled with ibconfig(). The contents of the status byte returned in *`result`* are device specific. Refer to the device manufacturer's documentation for details. For devices conforming to the IEEE488.1 or 2 specification the bits defined in the table below are available if enabled in the device's Status Byte Enable register.

Table 1. Standard IEEE.488 GPIB status byte bits

Return value

The value of ibsta is returned.

ibrsv

Name

`ibrsv` — request service (board)

Synopsis

```
#include <gpib/ib.h>
int ibrsv(int ud, int status_byte);
```

Description

The serial poll response byte of the board specified by the board descriptor *ud* is set to *status_byte*. If MSS (bit 6 in *status_byte*) is set, then the IEEE 488.2 local message "reqt" will be set true, causing the board to request service by asserting the SRQ line. If the MSS bit is clear, then the "reqf" message will be set true, causing the board to stop requesting service.

Boards will also automatically stop requesting service when they are serial polled by the controller.

This function follows the implementation technique described in IEEE 488.2 section 11.3.3.4.3. It is prone to generating spurious requests for service, which are permitted by 488.2 but less than ideal. In order to avoid spurious requests, use ibrsv2() instead.

Return value

The value of ibsta is returned.

ibrsv2

Name

`ibrsv2` — request service (board)

Synopsis

```
#include <gpib/ib.h>
int ibrsv2(int ud, int status_byte, int new_reason_for_request);
```

Description

The serial poll response byte of the board specified by the board descriptor *ud* is set to *status_byte*. A service request may be generated, cleared, or left unaffected depending on the values of MSS (bit 6 in *status_byte*) and *new_reason_for_request*.

There are three valid possibilities for MSS and *new_reason_for_request*. If MSS is 1 and *new_reason_for_request* is nonzero, then the IEEE 488.2 local message "reqt" will be set true. reqt sets local message "rsv" true which in turn causes the board to request service by asserting the SRQ line. If the MSS bit is 0 and *new_reason_for_request* is also 0, then the "reqf" message will be set true, causing rsv to clear and the board to stop requesting service. Finally, if MSS is 1 and *new_reason_for_request* is 0, then ibrsv2 will have no effect on the service request state (it will only update the status byte). The fourth possibilty of MSS is 0 (which implies no service request) and *new_reason_for_request* is nonzero (which implies there is a service request) is contradictory and will be rejected with an EARG error.

Boards will also automatically stop requesting service when they are serial polled by the controller.

This function follows the preferred implementation technique described in IEEE 488.2 section 11.3.3.4.1. It can be used to avoid the spurious requests for service that ibrsv() is prone to. However, not all drivers/hardware implement support for this function. In such a case, this function may result in a ECAP error, and you will have to fall back on using the simpler ibrsv().

If you are implementing a 488.2 device, this function should be called every time either the status byte changes, or the service request enable register changes. The value for *new_reason_for_request* may be calculated from:

```
new_reason_for_request = (status_byte & service_request_enable) &
                         ~(old_status_byte & old_service_request_enable);
```

Return value

The value of ibsta is returned.

ibsad

Name

`ibsad` — set secondary GPIB address (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibsad(int ud, int sad);
```

Description

ibsad() sets the GPIB secondary address of the device or board specified by the descriptor *ud*. If *ud* is a device descriptor, then the setting is local to the descriptor (it does not affect the behaviour of calls using other descriptors, even if they refer to the same physical device). If *ud* is a board descriptor, then the board's secondary address is changed immediately, which is a global change affecting anything (even other processes) using the board.

This library follows NI's unfortunate convention of adding 0x60 hexadecimal (96 decimal) to secondary addresses. That is, if you wish to set the secondary address to 3, you should set *sad* to 0x63. Setting *sad* to 0 disables the use of secondary addressing. Valid GPIB secondary addresses are in the range from 0 to 30 (which correspond to *sad* values of 0x60 to 0x7e).

Return value

The value of ibsta is returned.

ibsic

Name

`ibsic` — perform interface clear (board)

Synopsis

```
#include <gpib/ib.h>
```

```
int ibsic(int ud);
```

Description

ibsic() resets the GPIB bus by asserting the 'interface clear' (IFC) bus line for a duration of at least 100 microseconds. The board specified by *ud* must be the system controller in order to assert IFC. The interface clear causes all devices to untalk and unlisten, puts them into serial poll disabled state (don't worry, you will still be able to conduct serial polls), and the board becomes controller-in-charge.

Return value

The value of ibsta is returned.

ibspb

Name

`ibspb` — obtain length of serial poll bytes queue (device)

Synopsis

```
#include <gpib/ib.h>
int ibspb(int ud, short *result);
```

Description

ibspb() obtains the number of serial poll bytes queued for the device specified by *ud*. The number of queued serial poll bytes is stored in the location specified by *result*.

If automatic serial polling is enabled on the board controlling the device, the status byte is automatically read and queued whenever the device requests service. Automatic serial polling is controlled with ibconfig().

The queued status bytes are read with ibrsp().

Return value

The value of ibsta is returned.

ibsre

Name

`ibsre` — set remote enable (board)

Synopsis

```
#include <gpib/ib.h>
int ibsre(int ud, int enable);
```

Description

If *enable* is nonzero, then the board specified by the board descriptor *ud* asserts the REN line. If *enable* is zero, the REN line is unasserted. The board must be the system controller.

Return value

The value of ibsta is returned.

ibstop

Name

`ibstop` — abort asynchronous i/o operation (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibstop(int ud);
```

Description

ibstop() aborts an asynchronous i/o operation (for example, one started with ibcmda(), ibrda(), or ibwrta()).

The return value of ibstop() is counter-intuitive. On successfully aborting an asynchronous operation, the ERR bit is set in ibsta, and iberr is set to EABO. If the ERR bit is not set in ibsta, then there was no asynchronous i/o operation in progress. If the function failed, the ERR bit will be set and iberr will be set to some value other than EABO.

Return value

The value of ibsta is returned.

ibtmo

Name

`ibtmo` — adjust io timeout (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibtmo(int ud, int timeout);
```

Description

ibtmo() sets the timeout for IO operations and ibwait calls performed using the board or device descriptor *ud*. The actual amount of time before a timeout occurs may be greater than the period specified, but never less. *timeout* is specified by using one of the following constants:

Table 1. Timeout constants

Return value

The value of ibsta is returned.

ibtrg

Name

`ibtrg` — trigger device (device)

Synopsis

```
#include <gpib/ib.h>
int ibtrg(int ud);
```

Description

ibtrg() sends a GET (group execute trigger) command byte to the device specified by the device descriptor *ud*.

Return value

The value of ibsta is returned.

ibvers

Name

`ibvers` — Obtain the current linux gpib version.

Synopsis

```
#include <gpib/ib.h>
void ibvers(char ** version);
```

Description

ibvers() will return the current version string in *version*.

ibwait

Name

`ibwait` — wait for event (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibwait(int ud, int status_mask);
```

Description

ibwait() will sleep until one of the conditions specified in *status_mask* is true. The meaning of the bits in *status_mask* are the same as the bits of the ibsta status variable.

If *status_mask* is zero, then ibwait() will return immediately. This is useful if you simply wish to get an updated ibsta.

When calling ibwait() on a device, only the following condition bits in the *status_mask* are valid: TIMO, END, CMPL, and RQS. For the RQS bit to be set in the returned ibsta automatic serial polling must be enabled for the board controlling the device, see ibconfig(). The RQS condition is cleared by serial polling the device, see ibrsp().

If you wish to resynchronize and obtain the results from an asynchronous I/O operation, you must wait on CMPL by setting its bit in the `status_mask` parameter. Then if ibwait returns with CMPL set, it will have updated iberr, ibcnt, and the ERR bit of ibsta with the most recent asynchronous I/O results.

If TIMO is set in the `status_mask` parameter, then ibwait will timeout after the time period set by ibtmo and set TIMO in ibsta.

Return value

The value of ibsta is returned.

ibwrt

Name

`ibwrt` — write data bytes (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibwrt(int ud, const void *data, long num_bytes);
```

Description

ibwrt() is used to write data bytes to a device or board. The argument `ud` can be either a device or board descriptor. `num_bytes` specifies how many bytes are written from the user-supplied array `data`. EOI may be asserted with the last byte sent or when the end-of-string character is sent (see ibeos() and ibeot()). The write operation may be interrupted by a timeout (see ibtmo()), the board receiving a device clear command, or receiving an interface clear.

If `ud` is a device descriptor, then the library automatically handles addressing the device as listener and the interface board as talker, before sending the data bytes onto the bus.

If *ud* is a board descriptor, the board simply writes the data onto the bus. The controller-in-charge must address the board as talker.

After the ibwrt() call, ibcnt and ibcntl are set to the number of bytes written.

Return value

The value of ibsta is returned.

ibwrta

Name

`ibwrta` — write data bytes asynchronously (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibwrta(int ud, const void *buffer, long num_bytes);
```

Description

ibwrta() is similar to ibwrt() except it operates asynchronously. ibwrta() does not wait for the sending of the data bytes to complete, but rather returns immediately.

While an asynchronous operation is in progress, most library functions will fail with an EOIP error. In order to sucessfully complete an asynchronous operation, you must call ibwait() with CMPL set in the wait mask, until the CMPL bit is set ibsta. Asynchronous operations may also be aborted with an ibstop() or ibonl() call.

After the asynchronous I/O has completed and the results resynchronized with the current thread, the Linux-GPIB extensions AsyncIbsta, AsyncIberr, AsyncIbcnt and AsyncIbcntl may be useful to more cleanly separate the results of the asynchronous I/O from the results of the ibwait or similar call used to resynchronize.

Return value

The value of ibsta is returned.

ibwrtf

Name

`ibwrtf` — write data bytes from file (board or device)

Synopsis

```
#include <gpib/ib.h>
int ibwrtf(int ud, const char *file_path);
```

Description

ibwrtf() is similar to ibwrt() except that the data to be written is taken from a file instead of an array in memory. *file_path* specifies the file, which is written byte for byte onto the bus.

Return value

The value of ibsta is returned.

4.3. "Multidevice" API Functions

The "Multidevice" API functions provide similar functionality to the "Traditional" API functions. However, some of the "multidevice" functions can be performed on multiple devices simultaneously. For example, SendList() can be used to write a message to multiple devices. Such functions take an array of Addr4882_t as an argument. The end of the array is specified by setting the last element to the constant NOADDR.

AllSPoll

Name

`AllSPoll` — serial poll multiple devices

Synopsis

```
#include <gpib/ib.h>
void AllSPoll(int board_desc, Addr4882_t *addressList, short *resultList);
void AllSpoll(int board_desc, const Addr4882_t *addressList, short
*resultList);
```

Description

AllSPoll() causes the interface board specified by *board_desc* to serial poll all the GPIB addresses specified in the *addressList* array. The results of the serial polls are stored into *resultList*. If you only wish to serial poll a single device, ReadStatusByte() or ibrsp() may be more convenient.

This function may also be invoked with the alternate capitalization 'AllSpoll' for compatibility with NI's library.

DevClear

Name

`DevClear` — clear a device

Synopsis

```
#include <gpib/ib.h>
void DevClear(int board_desc, Addr4882_t address);
```

Description

DevClear() causes the interface board specified by *`board_desc`* to send the clear command to the GPIB addresses specified by *`address`*. The results of the serial polls are stored into *`resultList`*. If you wish to clear multiple devices simultaneously, use DevClearList()

DevClearList

Name

`DevClearList` — clear multiple devices

Synopsis

```
#include <gpib/ib.h>
void DevClearList(int board_desc, const Addr4882_t addressList[]);
```

Description

DevClear() causes the interface board specified by *`board_desc`* to send the clear command simultaneously to all the GPIB addresses specified by the *`addressList`* array. If *`addressList`* is empty or NULL, then the clear command is sent to all devices on the bus. If you only wish to clear a single device, DevClear() or ibclr() may be slightly more convenient.

EnableLocal

Name

`EnableLocal` — put devices into local mode.

Synopsis

```
#include <gpib/ib.h>
void EnableLocal(int board_desc, const Addr4882_t addressList[]);
```

Description

EnableLocal() addresses all of the devices in the `addressList` array as listeners then sends the GTL (go to local) command byte, causing them to enter local mode. This requires that the board is the controller-in-charge. Note that while the REN (remote enable) bus line is asserted, the devices will return to remote mode the next time they are addressed.

If `addressList` is empty or NULL, then the REN line is unasserted and all devices enter local mode. The board must be system controller to change the state of the REN line.

EnableRemote

Name

`EnableRemote` — put devices into remote mode.

Synopsis

```
#include <gpib/ib.h>
void EnableRemote(int board_desc, const Addr4882_t addressList[]);
```

Description

EnableRemote() asserts the REN (remote enable) line, and addresses all of the devices in the `addressList` array as listeners (causing them to enter remote mode). The board must be system controller.

FindLstn

Name

`FindLstn` — find devices

Synopsis

```
#include <gpib/ib.h>
```

```
void FindLstn(int board_desc, const Addr4882_t padList[], Addr4882_t
resultList[], int maxNumResults);
```

Description

FindLstn() will check the primary addresses in the `padList` array for devices. The GPIB addresses of all devices found will be stored in the `resultList` array, and ibcnt will be set to the number of devices found. The `maxNumResults` parameter limits the maximum number of results that will be returned, and is usually set to the number of elements in the `resultList` array. If more than `maxNumResults` devices are found, an ETAB error is returned in iberr. The `padList` should consist of primary addresses only, with no secondary addresses (all possible secondary addresses will be checked as necessary).

Your GPIB board must have the capability to monitor the NDAC bus line in order to use this function (see iblines).

This function has the additional effect of addressing the board as talker for the duration of the Find Listeners protocol, which is beyond what IEEE 488.2 specifies. This is done because some boards cannot reliably read the state of the NDAC bus line unless they are the talker. Being the talker causes the board's gpib transceiver to configure NDAC as an input, so its state can be reliably read from the bus through the transceiver.

FindRQS

Name

`FindRQS` — find device requesting service and read its status byte

Synopsis

```
#include <gpib/ib.h>
void FindRQS(int board_desc, const Addr4882_t addressList[], short *status);
```

Description

FindRQS will serial poll the GPIB addresses specified in the `addressList` array until it finds a device requesting service. The status byte of the device requesting service is stored in the location specified by `status`. The `addressList` array index of the device requesting service is returned in ibcnt. If no device requesting service is found, an ETAB error is returned in iberr.

PassControl

Name

`PassControl` — make device controller-in-charge

Synopsis

```
#include <gpib/ib.h>
void PassControl(int board_desc, const Addr4882_t address);
```

Description

PassControl() causes the board specified by *board_desc* to pass control to the device specified by *address*. On success, the device becomes the new controller-in-charge.

PPoll

Name

`PPoll` — parallel poll devices

Synopsis

```
#include <gpib/ib.h>
void PPoll(int board_desc, short *result);
```

Description

PPoll() is similar to the 'traditional' API function ibrpp(). It causes the interface board to perform a parallel poll, and stores the parallel poll byte in the location specified by *result*. Bits 0 to 7 of the parallel poll byte correspond to the dio lines 1 to 8, with a 1 indicating the corresponding dio line is asserted. The devices on the bus you wish to poll should be configured beforehand with PPollConfig(). The board must be controller-in-charge to perform a parallel poll.

PPollConfig

Name

`PPollConfig` — configure a device's parallel poll response

Synopsis

```
#include <gpib/ib.h>
void PPollConfig(int board_desc, Addr4882_t address, int dio_line, int
line_sense);
```

Description

PPollConfig() configures the device specified by *address* to respond to parallel polls. The *dio_line* (valid values are 1 through 8) specifies which dio line the device being configured should use to send back its parallel poll response. The *line_sense* argument specifies the polarity of the response. If *line_sense* is nonzero, then the specified dio line will be asserted to indicate that the 'individual status bit' (or 'ist') is 1. If *sense* is zero, then the specified dio line will be asserted when ist is zero.

PPollUnconfig

Name

`PPollUnconfig` — disable devices' parallel poll response

Synopsis

```
#include <gpib/ib.h>
void PPollUnconfig(int board_desc, const Addr4882_t addressList[]);
```

Description

PPollUnconfig() configures the devices specified by *addressList* to ignore parallel polls.

RcvRespMsg

Name

RcvRespMsg — read data

Synopsis

```
#include <gpib/ib.h>
void RcvRespMsg(int board_desc, void *buffer, long count, int termination);
```

Description

RcvRespMsg() reads data from the bus. A device must have already been addressed as talker (and the board as listener) before calling this function. Addressing may be accomplished with the ReceiveSetup() function.

Up to `count` bytes are read into the array specified by `buffer`. The `termination` argument specifies the 8-bit end-of-string character (which must be a value from 0 to 255) whose reception will terminate a read. `termination` can also be set to the 'STOPend' constant, in which case no end-of-string character will be used. Assertion of the EOI line will always end a read.

You may find it simpler to use the slightly higher level function Receive(), since it does not require addressing and reading of data to be performed separately.

ReadStatusByte

Name

ReadStatusByte — serial poll a device

Synopsis

```
#include <gpib/ib.h>
void ReadStatusByte(int board_desc, Addr4882_t address, short *result);
```

Description

ReadStatusByte() causes the board specified by the board descriptor `board_desc` to serial poll the GPIB address specified by `address`. The status byte is stored at the location specified by the `result` pointer. If you wish to serial poll multiple devices, it may be slightly more efficient to use AllSPoll(). Serial polls may also be conducted with the 'traditional API' function ibrsp().

Receive

Name

`Receive` — perform receive addressing and read data

Synopsis

```
#include <gpib/ib.h>
void Receive(int board_desc, Addr4882_t address, void *buffer, long count,
int termination);
```

Description

Receive() performs the necessary addressing, then reads data from the device specified by `address`. It is equivalent to a ReceiveSetup() call followed by a RcvRespMsg() call.

ReceiveSetup

Name

`ReceiveSetup` — perform receive addressing

Synopsis

```
#include <gpib/ib.h>
void ReceiveSetup(int board_desc, Addr4882_t address);
```

Description

ReceiveSetup() addresses the device specified by *address* as talker, and addresses the interface board as listener. A subsequent RcvRespMsg() call will read data from the device.

You may find it simpler to use the slightly higher level function Receive(), since it does not require addressing and reading of data to be performed separately.

ResetSys

Name

`ResetSys` — reset system

Synopsis

```
#include <gpib/ib.h>
void ResetSys(int board_desc, const Addr4882_t addressList[]);
```

Description

ResetSys() has the following effects:

- The remote enable bus line is asserted.
- An interface clear is performed (the interface clear bus line is asserted for at least 100 microseconds).
- The device clear command is sent to all the devices on the bus.
- The *RST message is sent to every device specified in the *addressList*.

Send

Name

`Send` — perform send addressing and write data

Synopsis

```
#include <gpib/ib.h>
void Send(int board_desc, Addr4882_t address, const void *data, long count,
int eot_mode);
```

Description

Send() addresses the device specified by `address` as listener, then writes data onto the bus. It is equivalent to a SendList() except it only uses a single GPIB address to specify the listener instead of allowing an array of listeners.

SendCmds

Name

`SendCmds` — write command bytes onto bus

Synopsis

```
#include <gpib/ib.h>
void SendCmds(int board_desc, const void *cmds, long count);
```

Description

SendCmds() writes `count` command byte onto the the GPIB bus from the array `cmds`.

It is generally not necessary to call SendCmds(). It is provided for advanced users who want direct, low-level access to the GPIB bus.

SendDataBytes

Name

`SendDataBytes` — write data

Synopsis

```
#include <gpib/ib.h>
void SendDataBytes(int board_desc, const void *data, long count, int
eot_mode);
```

Description

SendDataBytes() writes data to the bus. One or more devices must have already been addressed as listener (and the board as talker) before calling this function. Addressing may be accomplished with the SendSetup() function.

`count` bytes are written from the array specified by `data`. The `eot_mode` argument specifies how the message should be terminated, and may be any of the following values:

Table 1. eot modes

You may find it simpler to use the slightly higher level functions Send() or SendList(), since they does not require addressing and writing of data to be performed separately.

SendIFC

Name

`SendIFC` — perform interface clear

Synopsis

```
#include <gpib/ib.h>
void SendIFC(int board_desc);
```

Description

SendIFC() resets the GPIB bus by asserting the 'interface clear' (IFC) bus line for a duration of at least 100 microseconds. The board specified by *`board_desc`* must be the system controller in order to assert IFC. The interface clear causes all devices to untalk and unlisten, puts them into serial poll disabled state (don't worry, you will still be able to conduct serial polls), and the board becomes controller-in-charge.

SendList

Name

`SendList` — write data to multiple devices

Synopsis

```
#include <gpib/ib.h>
void SendList(int board_desc, const Addr4882_t addressList[], const void
*data, long count, int eot_mode);
```

Description

SendList() addresses the devices in *`addressList`* as listeners, then writes the contents of the array *`data`* to them. It is equivalent to a SendSetup() call followed by a SendDataBytes() call.

SendLLO

Name

`SendLLO` — put devices into local lockout mode

Synopsis

```
#include <gpib/ib.h>
void SendLLO(int board_desc);
```

Description

SendLLO() asserts the 'remote enable' bus line, then sends the LLO command byte. Any devices currently addressed as listener will be put into RWLS (remote with lockout state), and all other devices will enter LWLS (local with lockout state). Local lockout means the remote/local mode of devices cannot be changed though the devices' front-panel controls. Unasserting the REN line should bring the devices out of lockout state.

The SetRWLS() performs a similar function, except it lets you specifiy which devices you wish to address as listener before sending the LLO command.

SendSetup

Name

`SendSetup` — perform send addressing

Synopsis

```
#include <gpib/ib.h>
void SendSetup(int board_desc, const Addr4882_t addressList[]);
```

Description

SendSetup() addresses the devices in *`addressList`* as listeners, and addresses the interface board as talker. A subsequent SendDataBytes() call will write data to the devices.

You may find it simpler to use the slightly higher level functions Send() or SendList(), since they does not require addressing and writing of data to be performed separately.

SetRWLS

Name

`SetRWLS` — put devices into remote with lockout state

Synopsis

```
#include <gpib/ib.h>
void SetRWLS(int board_desc, const Addr4882_t addressList[]);
```

Description

SetRWLS() asserts the 'remote enable' bus line, addresses the devices in the *addressList* array as listeners, then sends the LLO command byte. The devices addressed as listener will be put into RWLS (remote with lockout state), and all other devices will enter LWLS (local with lockout state). Local lockout means the remote/local mode of devices cannot be changed though the devices' front-panel controls. Unasserting the REN line should bring the devices out of the lockout state.

TestSRQ

Name

`TestSRQ` — query state of SRQ bus line

Synopsis

```
#include <gpib/ib.h>
void TestSRQ(int board_desc, short *result);
```

Description

TestSRQ() checks the state of the SRQ bus line and writes its state to the location specified by *result*. A '1' indicates the SRQ line is asserted, and a '0' indicates the line is not asserted.

Some boards lack the capability to report the status of the SRQ line. In such a case, an ECAP error is returned in iberr.

TestSys

Name

`TestSys` — perform self-test queries on devices

Synopsis

```
#include <gpib/ib.h>
void TestSys(int board_desc, const Addr4882_t addressList[], short
results[]);
```

Description

TestSys() sends the '*TST?' message to all the devices in the `addressList` array, then reads their responses into the `results` array. This will cause devices that conform to the IEEE 488.2 standard to perform a self-test and respond with a zero on success. A non-zero response indicates an error during the self-test.

The number of devices which responded with nonzero values from their self-tests is returned in ibcnt and ibcntl. If a device fails to respond to the *TST? query, an error will be flagged in ibsta (this is different than NI's documented behaviour which is broken).

Trigger

Name

`Trigger` — trigger a device

Synopsis

```
#include <gpib/ib.h>
void Trigger(int board_desc, Addr4882_t address);
```

Description

Trigger() is equivalent to a TriggerList() call with a single address.

TriggerList

Name

`Trigger` — trigger multiple devices

Synopsis

```
#include <gpib/ib.h>
void TriggerList(int board_desc, Addr4882_t addressList[]);
```

Description

TriggerList() sends a GET (group execute trigger) command byte to all the devices specified in the *addressList* array. If no addresses are specified in *addressList* then the GET command byte is sent without performing any addressing.

WaitSRQ

Name

`WaitSRQ` — sleep until the SRQ bus line is asserted

Synopsis

```
#include <gpib/ib.h>
void WaitSRQ(int board_desc, short *result);
```

Description

WaitSRQ() sleeps until either the SRQ bus line is asserted, or a timeout (see ibtmo()) occurs. A '1' will be written to the location specified by *result* if SRQ was asserted, and a '0' will be written if the function timed out.

4.4. Utility Functions

AsyncIbcnt and AsyncIbcntl

Name

`AsyncIbcnt and AsyncIbcntl` — ibcnt and ibcntl values for last asynchronous I/O operation

Synopsis

```
#include <gpib/ib.h>
int AsyncIbcnt(void);
long AsyncIbcntl(void);
```

Description

AsyncIbcnt() and AsyncIbcntl() return thread-local counts related to the global variables ibcnt and ibcntl. Their values correspond to the result of the last asynchronous I/O operation resynchronized to the current thread by an ibwait or ibstop call. These functions only reflect the result of the asynchronous I/O operation itself and not, for example, the ibwait which resynchronized the asynchronous result to the current thread. Thus the result from AsyncIbcnt() is easier to interpret than ThreadIbcnt(), since it is unambiguous whether the value is associated with the asynchronous I/O result, or with the function call used to resynchronize (ibwait or ibstop).

These functions are Linux-GPIB extensions.

Return value

A value related to ibcnt or ibcntl corresponding to the last asynchronous I/O operation resynchronized to the current thread is returned.

AsyncIberr

Name

`AsyncIberr` — iberr value for last asynchronous I/O operation

Synopsis

```
#include <gpib/ib.h>
int AsyncIberr(void);
```

Description

AsyncIberr() returns a thread-local error number related to the global variable iberr. Its value corresponds to the result of the last asynchronous I/O operation resynchronized to the current thread by an ibwait or ibstop call. This function only reflects the result of the asynchronous I/O operation itself and not, for example, the ibwait which resynchronized the asynchronous result to the current thread. Thus the result from AsyncIberr() is easier to interpret than ThreadIberr(), since it is unambiguous whether the value is associated with the asynchronous I/O result, or with the function call used to resynchronize (ibwait or ibstop).

This function is a Linux-GPIB extension.

Return value

A value related to iberr corresponding to the last asynchronous I/O operation resynchronized to the current thread is returned.

AsyncIbsta

Name

`AsyncIbsta` — ibsta value for last asynchronous I/O operation

Synopsis

```
#include <gpib/ib.h>
int AsyncIbsta(void);
```

Description

AsyncIbsta() returns a thread-local status value related to the global variable ibsta. Its value corresponds to the result of the last asynchronous I/O operation resynchronized to the current thread by an ibwait or ibstop call. This function only reflects the result of the asynchronous I/O operation itself and not, for example, the ibwait which resynchronized the asynchronous result to the current thread. Thus the result from AsyncIbsta() is easier to interpret than ThreadIbsta(), since it is unambiguous whether the value is associated with the asynchronous I/O result, or with the function call used to resynchronize (ibwait or ibstop).

Only the status bits END | ERR | TIMO | CMPL are valid in the returned status byte. The rest of the bits should be ignored and will be set to zero.

This function is a Linux-GPIB extension.

Return value

A value related to ibsta corresponding to the last asynchronous I/O operation resynchronized to the current thread.

CFGn

Name

`CFGn` — generate ’configure n meters’ command byte

Synopsis

```
#include <gpib/ib.h>
uint8_t CFGn(unsigned int num_meters);
```

Description

CFGn() returns a ’configure n meters’ command byte corresponding to the *`num_meters`* argument. *`num_meters`* (valid values are 1 through 15) specifies how many meters of cable are in your system. This is necessary in before high speed non-interlocked handshaking (a.k.a. HS488) can be used on the bus. The CFGn command byte must be preceded by a CFE command byte to take effect.

Return value

The appropriate CFGn command byte is returned.

GetPAD

Name

`GetPAD` — extract primary address from an Addr4882_t value

Synopsis

```
#include <gpib/ib.h>
static __inline__ unsigned int GetPAD(Addr4882_t address);
```

Description

GetPAD() extracts the primary address packed into the Addr4882_t value *address*.

Return value

The primary GPIB address (from 0 through 30) stored in *address*.

GetSAD

Name

`GetSAD` — extract secondary address from an Addr4882_t value

Synopsis

```
#include <gpib/ib.h>
static __inline__ unsigned int GetSAD(Addr4882_t address);
```

Description

GetSAD() extracts the secondary address packed into the Addr4882_t value *address*.

Return value

The secondary GPIB address (from 0x60 through 0x7e, or 0 for none) stored in *address*.

MakeAddr

Name

`MakeAddr` — pack primary and secondary address into an Addr4882_t value

Synopsis

```
#include <gpib/ib.h>
static __inline__ Addr4882_t MakeAddr(unsigned int pad, unsigned int sad);
```

Description

MakeAddr() generates an Addr4882_t value that corresponds to the specified primary address *pad* and secondary address *sad*. It does so by putting *pad* into the least significant byte and left shifting *sad* up to the next byte.

Examples

```
Addr4882_t addressList[ 5 ];

addressList[ 0 ] = 5 /* primary address 5, no secondary address */
addressList[ 1 ] = MakeAddr(3, 0); /* primary address 3, no secondary address */
addressList[ 2 ] = MakeAddr(7, 0x70); /* primary address 3, secondary address 16 */
addressList[ 3 ] = MakeAddr(20, MSA(9)); /* primary address 20, secondary address 9 */
addressList[ 4 ] = NOADDR;
```

Return value

An Addr4882_t value corresponding to the specified primary and secondary GPIB address.

MLA

Name

`MLA` — generate 'my listen address' command byte

Synopsis

```
#include <gpib/ib.h>
uint8_t MLA(unsigned int address);
```

Description

MLA() returns a 'my listen address' command byte corresponding to the *address* argument. The *address* may be between 0 and 30.

Return value

The appropriate MLA command byte is returned.

MSA

Name

`MSA` — generate 'my secondary address' command byte

Synopsis

```
#include <gpib/ib.h>
uint8_t MSA(unsigned int address);
```

Description

MSA() returns a 'my secondary address' command byte corresponding to the *address* argument. The *address* may be between 0 and 30. This macro is also useful for mangling secondary addresses from the 'real' values between 0 and 30 to the range 0x60 to 0x7e used by most of the library's functions.

Return value

The appropriate MSA command byte is returned.

MTA

Name

MTA — generate 'my talk address' command byte

Synopsis

```
#include <gpib/ib.h>
uint8_t MTA(unsigned int address);
```

Description

MTA() returns a 'my talk address' command byte corresponding to the *address* argument. The *address* may be between 0 and 30.

Return value

The appropriate MTA command byte is returned.

PPE_byte

Name

PPE_byte — generate ’parallel poll enable’ command byte

Synopsis

```
#include <gpib/ib.h>
uint8_t PPE_byte(unsigned int dio_line, int sense);
```

Description

PPE_byte() returns a ’parallel poll enable’ command byte corresponding to the *dio_line* and *sense* arguments. The *dio_line* (valid values are 1 through 8) specifies which dio line the device being configured should use to send back its parallel poll response. The *sense* argument specifies the polarity of the response. If *sense* is nonzero, then the specified dio line will be asserted to indicate that the ’individual status bit’ (or ’ist’) is 1. If *sense* is zero, then the specified dio line will be asserted when ist is zero.

Return value

The appropriate PPE command byte is returned.

ThreadIbcnt and ThreadIbcntl

Name

ThreadIbcnt and ThreadIbcntl — thread-specific ibcnt and ibcntl values

Synopsis

```
#include <gpib/ib.h>
int ThreadIbcnt(void);
long ThreadIbcntl(void);
```

Description

ThreadIbcnt() and ThreadIbcntl() return thread-local versions of the global variables ibcnt and ibcntl.

Return value

The value of ibcnt or ibcntl corresponding to the last 'traditional' or 'multidevice' function called in the current thread is returned.

ThreadIberr

Name

`ThreadIberr` — thread-specific iberr value

Synopsis

```
#include <gpib/ib.h>
int ThreadIberr(void);
```

Description

ThreadIberr() returns a thread-local version of the global variable iberr.

Return value

The value of iberr corresponding to the last 'traditional' or 'multidevice' function called by the current thread is returned.

ThreadIbsta

Name

`ThreadIbsta` — thread-specific ibsta value

Synopsis

```
#include <gpib/ib.h>
int ThreadIbsta(void);
```

Description

ThreadIbsta() returns a thread-local version of the global variable ibsta.

Return value

The value of ibsta corresponding to the last 'traditional' or 'multidevice' function called by the current thread is returned.

5. GPIB protocol

5.1. GPIB command bytes

The meaning and values of the possible GPIB command bytes are as follows:

Table 13. GPIB command bytes

5.2. GPIB bus lines

Physically, the GPIB bus consists of 8 data lines, 3 handshaking lines, and 5 control lines (and 8 ground lines). Brief descriptions of how they are used follow:

Table 14. GPIB bus lines

A. GNU Free Documentation License

A.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the

Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

A.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

A.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.12. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.