Supplement to the NI-488M Software Reference Manual for AIX

This supplement contains a discussion of the NI-488 function `ibdev` and the AIX-specific functions `ibhwdiag` and `ibpost` that are not documented in the standard *NI-488M Software Reference Manual*. The purpose, syntax, and C language programming examples for each function are given. The functions `ibdev` and `ibpost` are part of the standard AIX GPIB language interface. The `ibhwdiag` function is available as a separate linkable object file, which is not included with the AIX driver software. For descriptions of all other NI-488 functions, please refer to the *NI-488M Software Reference Manual*.

NI-488™ and NI-488M™ are trademarks of National Instruments Corporation. Product and company names are trademarks or trade names of their respective companies.

321319A-01 *© Copyright 1996 National Instruments Corp. All rights reserved.* *June 1996*

Name

`ibdev` - open and initialize an unused device when the device name is unknown

Synopsis

```
#include <sys/ugpib.h>
ud = int ibdev (int boardindex, int pad, int sad,
                int tmo, int eot, int eos)
```

Description

`boardindex` is an index from 0 to [(number of boards) - 1] of the access board that the device descriptor must be associated with. The arguments `pad`, `sad`, `tmo`, `eot`, and `eos` dynamically set the software configuration for the NI-488 I/O functions. These arguments configure the primary address, secondary address, I/O timeout, asserting EOI on last byte of data sourced, and the End-Of-String mode and byte, respectively. (Refer to the *ibpad*, *ibsad*, *ibtmo*, *ibeot*, and *ibeos* function descriptions for more information on each argument.) The device descriptor is returned in the variable `ud`.

The `ibdev` command selects an unopened device, opens it, and initializes it. You can use this function in place of `ibfind`.

`ibdev` returns a device descriptor of the first unopened user-configurable device that it finds. For this reason, it is very important to use `ibdev` *only after* all of your `ibfind` calls have been made. This is the only way to ensure that `ibdev` does not use a device that you plan to use via an `ibfind` call. The `ibdev` function performs the equivalent of the `ibonl` function to open the device.

Note: ***The device descriptor of the NI-488M driver can remain open across invocations of an application, so be sure to return the device descriptor to the pool of available devices by calling*** `ibonl` ***with*** `v=0` ***when you are finished using the device. If you do not, that device will not be available for the next*** `ibdev` ***call.***

If the `ibdev` call fails, a negative number is returned in place of the device descriptor. There are two distinct errors that can occur with the `ibdev` call:

- If no device is available or the specified board index refers to a non-existent board, it returns the EDVR or ENEB error.

- If one of the last five parameters is an illegal value, it returns with a good board descriptor and the EARG error.

Example

`ibdev` opens an available device and assigns it to access `gpib0` (`board` = 0) with a primary address of 6 (`pad` = 6), a secondary address of 0x67 (`sad` = 0x67), a timeout of 10 ms (`tmo` = 7), the END message enabled (`eot` = 1) and the EOS mode disabled (`eos` = 0).

```
if ((ud = ibdev(0,6,0x67,7,1,0)) < 0) {
     /* Handle GPIB error here */
     if (iberr == EDVR) {
          /* bad boardindex or no devices
           * available.
           */
     }
     else if (iberr == EARG) {
          /* The call succeeded, but at least one
           * of pad, sad, tmo, eos, eot is incorrect.
           */
     }
```

See Also

ibfind (3), *ibpad (3)*, *ibsad (3)*, *ibtmo (3)*, *ibeot (3)*, and *ibeos (3)* in the *NI-488M Software Reference Manual*

Name

`ibhwdiag` - perform hardware diagnostic tests on the specified board

Synopsis

`ibhwdiag(`int `boardnum,` int `cable)`

Description

`boardnum` is an index from 0 to [(number of boards) - 1]. `cable` should be set to zero if no cable is attached and all tests can be performed, or 1 if a cable is attached. If a cable is attached, only the tests that can work with a cable attached to the MC-GPIB are performed.

`ibhwdiag` performs a series of tests on the GPIB hardware, where each test is uniquely identified by a number. The `ibhwdiag` function performs these hardware tests by accessing the GPIB driver.

The `ibhwdiag` function returns -1 if there is a communication problem with the GPIB driver, and -2 if the board is not installed. `ibhwdiag` returns 0 if all the tests completed successfully. If one of the hardware tests failed, the function returns a value greater than 0 with the return value corresponding to the test number that failed.

Examples

1. Run hardware diagnostics on `gpib0` with cable attached.

```
ibhwdiag(0,1);
```

2. Run hardware diagnostics on `gpib2` with no cables attached.

```
ibhwdiag(2,0);
```

See Also

The *ibdiag* section in the *Getting Started* manual

Name

`ibpost` - request notification of the occurrence of an SRQ event

Synopsis

```
#include <sys/ugpib.h>
int ibpost(int ud, int eventmask)
```

Description

`ud` designates a board descriptor, and `eventmask` specifies a mask bit you want to associate with an SRQ event. There is no defined mask value to represent the GPIB SRQ event. In applications that use `ibpost`, you must define your own mask value to represent an SRQ event. However, you need to make sure that the value you define does not coincide with the predefined mask values the system reserves to represent certain general events. Please refer to the system file `sleep.h` for a list of the mask values defined for general events.

Like the NI-488 function `ibsgnl`, `ibpost` returns immediately, freeing the application program to perform other tasks. When an SRQ interrupt occurs, the driver issues an `e_post` with the SRQ `eventmask` as one of its parameters. This causes the kernel to set the SRQ mask bit. The only way the application can examine and clear this bit is with the `e_wait` kernel function. Therefore, when the process needs to check or wait for an occurrence of the SRQ interrupt, it has to execute an `e_wait` on the SRQ mask bit.

Note: ***To use this function you must have access rights to use the kernel extensions from your application.***

Example

Call `ibpost` with the SRQ event bit defined to be 0x2000000.

```
ibpost(brd0, 0x2000000);
```

See Also

The *e_post* and *e_wait* kernel function descriptions on the `man` pages