NAME

devctl, **devctl_attach**, **devctl_clear_driver**, **devctl_delete**, **devctl_detach**, **devctl_disable**, **devctl_enable**, **devctl_freeze**, **devctl_getpath**, **devctl_rescan**, **devctl_reset**, **devctl_resume**, **devctl_set_driver**, **devctl_suspend**, **devctl_thaw** **-** device control library

LIBRARY

Device Control Library (libdevctl, -ldevctl)

SYNOPSIS

#include <devctl.h>

int
devctl_attach(*const char *device*);

int
devctl_clear_driver(*const char *device, bool force*);

int
devctl_delete(*const char *device, bool force*);

int
devctl_detach(*const char *device, bool force*);

int
devctl_disable(*const char *device*, *bool force_detach*);

int
devctl_enable(*const char *device*);

int
devctl_freeze(*void*);

int
devctl_getpath(*const char *device*, *const char *locator*, *char **buffer*);

int
devctl_rescan(*const char *device*);

int
devctl_reset(*const char *device, bool detach*);

int
devctl_resume(*const char *device*);

int
devctl_set_driver(*const char *device*, *const char *driver*, *bool force*);

int
devctl_suspend(*const char *device*);

int
devctl_thaw(*void*);

DESCRIPTION

The **devctl** library adjusts the state of devices in the kernel's internal device hierarchy. Each control operation accepts a *device* argument that identifies the device to adjust. The *device* may be specified as either the name of an existing device or as a bus-specific address. The following bus-specific address formats are currently supported:

pci*domain*:*bus*:*slot*:*function*
A PCI device with the specified *domain*, *bus*, *slot*, and *function*.

pci*bus*:*slot*:*function*
A PCI device in domain zero with the specified *bus*, *slot*, and *function*.

handle
A device with an ACPI handle of *handle*. The handle must be specified as an absolute path and must begin with a "\".

The **devctl_attach**() function probes a device and attaches a suitable device driver if one is found.

The **devctl_detach**() function detaches a device from its current device driver. The device is left detached until either a new driver for its parent bus is loaded or the device is explicitly probed via **devctl_attach**(). If *force* is true, the current device driver will be detached even if the device is busy.

The **devctl_delete**() function deletes a device from the device tree. No If *force* is true, the device is deleted even if the device is physically present.

The **devctl_disable**() function disables a device. If the device is currently attached to a device driver, the device driver will be detached from the device, but the device will retain its current name. If *force_detach* is true, the current device driver will be detached even if the device is busy. The device will remain disabled and detached until it is explicitly enabled via **devctl_enable**().

The **devctl_enable**() function re-enables a disabled device. The device will probe and attach if a suitable device driver is found.

The **devctl_suspend**() function suspends a device. This may include placing the device in a reduced power state, but any device driver currently attached to the device will remain attached.

The **devctl_resume**() function resumes a suspended device to a fully working state.

The **devctl_set_driver**() function attaches a device driver named *driver* to a device. If the device is already attached and *force* is false, the request will fail. If the device is already attached and *force* is true, the device will be detached from its current device driver before it is attached to the new device driver.

The **devctl_clear_driver**() function resets a device so that it can be attached to any valid device driver rather than only drivers with a previously specified name. This function is used to undo a previous call to **devctl_set_driver**(). If the device is already attached and *force* is false, the request will fail. If the device is already attached and *force* is true, the device will be detached from its current device driver. After the device's name is reset, it is reprobed and attached to a suitable device driver if one is found.

The **devctl_rescan**() function rescans a bus device checking for devices that have been added or removed.

The **devctl_getpath**() retrieves the path to the *device* from the kernel using the *locator* method to construct the path. The *buffer* pointer is updated with an allocated buffer that must be freed with free.

The **devctl_freeze**() function freezes probe and attach processing initiated in response to drivers being loaded.

The **devctl_thaw**() function resumes (thaws the freeze) probe and attach processing initiated in response to drivers being loaded.

The **devctl_reset**() function resets the specified device using bus-specific reset method. The *detach* argument, if true, specifies that the device driver is detached before the reset, and re-attached afterwards. If false, the device is suspended before the reset, and resumed after.

RETURN VALUES

The **devctl_attach**(), **devctl_clear_driver**(), **devctl_delete**(), **devctl_detach**(), **devctl_disable**(), **devctl_enable**(), **devctl_suspend**(), **devctl_rescan**(), **devctl_resume**(), and **devctl_set_driver**() functions return the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

ERRORS

In addition to specific errors noted below, all of the **devctl** functions may fail for any of the errors described in open(2) as well as:

[EINVAL]	The device name is too long.
[ENOENT]	No existing device matches the specified name or location.
[EPERM]	The current process is not permitted to adjust the state of *device*.

The **devctl_attach**() function may fail if:

[EBUSY]	The device is already attached.
[ENOMEM]	An internal memory allocation request failed.
[ENXIO]	The device is disabled.
[ENXIO]	No suitable driver for the device could be found, or the driver failed to attach.

The **devctl_detach**() function may fail if:

[EBUSY]	The current device driver for *device* is busy and cannot detach at this time. Note that some drivers may return this even if *force* is true.
[ENXIO]	The device is not attached to a driver.
[ENXIO]	The current device driver for *device* does not support detaching.

The **devctl_enable**() function may fail if:

[EBUSY]	The device is already enabled.
[ENOMEM]	An internal memory allocation request failed.
[ENXIO]	No suitable driver for the device could be found, or the driver failed to attach.

The **devctl_disable**() function may fail if:

[EBUSY]	The current device driver for *device* is busy and cannot detach at this time. Note that some drivers may return this even if *force_detach* is true.
[ENXIO]	The device is already disabled.
[ENXIO]	The current device driver for *device* does not support detaching.

The **devctl_suspend**() function may fail if:

[EBUSY]	The device is already suspended.
[EINVAL]	The device to be suspended is the root bus device.

The **devctl_resume**() function may fail if:

[EINVAL]	The device is not suspended.
[EINVAL]	The device to be resumed is the root bus device.

The **devctl_set_driver**() function may fail if:

[EBUSY]	The device is currently attached to a device driver and *force* is false.
[EBUSY]	The current device driver for *device* is busy and cannot detach at this time.
[EFAULT]	The *driver* argument points outside the process' allocated address space.
[ENOENT]	No device driver with the requested name exists.
[ENOMEM]	An internal memory allocation request failed.
[ENXIO]	The device is disabled.
[ENXIO]	The new device driver failed to attach.

The **devctl_clear_driver**() function may fail if:

[EBUSY]	The device is currently attached to a device driver and *force* is false.

[EBUSY]	The current device driver for *device* is busy and cannot detach at this time.
[EINVAL]	The device is not configured for a specific device driver name.
[ENXIO]	The device driver chosen after reprobing failed to attach.

The **devctl_rescan**() function may fail if:

[ENXIO]	The device is not attached to a driver.
[ENXIO]	The bus driver does not support rescanning.

The **devctl_delete**() function may fail if:

[EBUSY]	The device is physically present and *force* is false.
[EINVAL]	*dev* is the root device of the device tree.

The **devctl_reset**() function may fail if:

[ENXIO]	The bus does not implement the reset method.
[ETIMEDOUT]	The device failed to respond after the reset in the time limits specific to the bus.

The **devctl_reset**() function may also return errors caused by the attach, detach, suspend, and resume methods of the device driver.

SEE ALSO

devinfo(3), devstat(3), devctl(8)

HISTORY

The **devctl** library first appeared in FreeBSD 10.3.

BUGS

If a device is suspended individually via **devctl_suspend**() and the entire machine is subsequently suspended, the device will be resumed when the machine resumes.

Similarly, if the device is suspended, and **devctl_reset**() is called on the device with *detach* set to *false*, the device is resumed by the **devctl_reset**() call. Or, if the driver for the device is detached manually, and **devctl_reset**() is called on the device with *detach* set to *true*, device reset re-attaches the driver.