NAME

ddi_dma_cookie_get, **ddi_dma_cookie_iter**, **ddi_dma_cookie_one**, **ddi_dma_ncookies**, **ddi_dma_nextcookie** - retrieve DMA cookies

SYNOPSIS

#include <sys/ddi.h>
#include <sys/sunddi.h>

*const ddi_dma_cookie_t **
ddi_dma_cookie_iter(*ddi_dma_handle_t handle*, *const ddi_dma_cookie_t *cookiep*);

*const ddi_dma_cookie_t **
ddi_dma_cookie_get(*ddi_dma_handle_t handle*, *uint_t index*);

*const ddi_dma_cookie_t **
ddi_dma_cookie_one(*ddi_dma_handle_t handle*);

uint_t
ddi_dma_ncookies(*ddi_dma_handle_t handle*);

void
ddi_dma_nextcookie(*ddi_dma_handle_t handle*, *ddi_dma_cookie_t *cookiep*);

PARAMETERS

handle	The DMA handle obtained by a call to ddi_dma_alloc_handle(9F).
cookie	A pointer to a ddi_dma_cookie(9S) structure.
index	An unsigned integer that represents the index of a cookie to obtain. The first entry is at index zero.

DESCRIPTION

The **ddi_dma_cookie_iter**(), **ddi_dma_cookie_get**(), and **ddi_dma_cookie_one**() functions obtain information about DMA cookies. When a DMA request, represented by the DMA handle *handle*, has been bound to a series of addresses with the ddi_dma_addr_bind_handle(9F) or ddi_dma_buf_bind_handle(9F) functions, the resulting addresses are stored in one or more ddi_dma_cookie(9S) structures. the three different functions provide different ways to obtain cookies and are safe alternatives to the unsafe **ddi_dma_nextcookie**() function. To see how to use these functions, please see the *EXAMPLES* section.

The **ddi_dma_cookie_iter**() function provides a way to iterate over all the cookies that are associated with the DMA handle *handle*. To get the first handle, pass NULL in *cookiep*. Do not use the DMA cookie returned from either of the ddi_dma_addr_bind_handle(9F) or ddi_dma_buf_bind_handle(9F) functions. To get subsequent cookies, pass the returned cookie as the argument *cookiep*. When the function returns NULL then that indicates that the last handle has been iterated over.

The **ddi_dma_cookie_get**() function returns a specific cookie. The *index* indicates which of the cookies should be returned. The first cookie is at index **0**. If an invalid index is specified, the function returns NULL.

The *ddi_da_cookie_one* function is a convenience function for DMA requests that have a single cookie. This function always returns the single cookie assosciated with the DMA handle *handle*. If this function is used when there is a DMA request with multiple cookies, then it will return NULL. Violating this may trigger an assertion on **DEBUG** kernels.

The **ddi_dma_ncookies**() function returns the number of DMA cookies that are associated with the DMA handle *handle*. If there are no DMA resources bound to the handle, then this will return **0**.

The **ddi_dma_nextcookie**() function was the historical function that was associated with obtaining DMA cookies. It should not be used due to several flaws. The **ddi_dma_nextcookie**() function mutates the underlying DMA handle meaning that a driver cannot obtain a cookie a second time and thus a device driver using this interface must either manually keep storage of the cookie around wasting space or rebind the handle, wasting time. In addition, there is no way for the function to indicate that a driver has consumed all of its cookies. If for some reason a device driver calls the **ddi_dma_nextcookie**() function more times than there are cookies, the results are undefined. In short, this function should not be used for any purpose. Use the **ddi_dma_cookie_iter**(), **ddi_dma_cookie_get**(), or **ddi_dma_cookie_one**() functions instead.

CONTEXT

The **ddi_dma_cookie_iter**(), **ddi_dma_cookie_get**(), **ddi_dma_cookie_one**(), **ddi_dma_ncookies**(), and **ddi_dma_nextcookie**() functions may be called from **user**, **kernel**, or **interrupt** context.

RETURN VALUES

Upon successful completion, the **ddi_dma_cookie_iter**(), **ddi_dma_cookie_get**(), **ddi_dma_cookie_one**() functions will return the requested DMA cookie. If there are no more cookies, or *coookiep* is invalid, the **ddi_dma_cookie_iter**() function will return NULL. If *index* does not correspond to a valid cookie, the **ddi_dma_cookie_get**() function will return NULL. If there is more than one cookie or another error occurs, the **ddi_dma_cookie_one**() function will return NULL.

Upon successful completion, the **ddi_dma_ncookies**() function returns the number of cookies associated with *handle*. If there are none, then **0** is returned.

The **ddi_dma_nextcookie**() function always updates *cookiep* regardless of whether it is valid or not.

EXAMPLES

Example 1 Using the **ddi_dma_cookie_iter**() function to obtain all DMA cookies.

```
/*
 * This example assumes that either ddi_dma_addr_bind_handle() or
 * ddi_dma_buf_bind_handle() has already been successfully called.
 */
void
program_dma(ddi_dma_handle_t handle)
{
        const ddi_dma_cookie_t *c;

        for (cookie = ddi_dma_cookie_iter(handle, NULL); c != NULL;
            c = ddi_dma_cookie_iter(handle, c)) {
                /*
                 * Use the dmac_laddress and dmac_size members to
                 * properly program the device or descriptor rings.
                 */
        }
}
```

Example 2 Using the **ddi_dma_cookie_get**() function.

```
/*
 * This example assumes that either ddi_dma_mem_alloc() has already
 * been successfully called.
 */
int
bind_dma(ddi_dma_handle_t handle, void *addr, size_t len)
{
        int ret;
        uint_t i, ncookies;
        ddi_dma_cookie_t first;

        ret = ddi_dma_addr_bind_handle(handle, NULL, addr, len,
            DDI_DMA_RDWR | DDI_DMA_CONSISTENT, NULL, NULL, &first,
            &ncookies);
        if (ret != DDI_DMA_MAPPED) {
                return (ret);
        }

        /*
         * A driver doesn't need to store ncookies. It can get it again
         * by simply calling ddi_dma_ncookies() and using the result in
         * place of ncookies from ddi_dma_addr_bind_handle().
         */
        for (i = 0; i < ncookies; i++) {
                const ddi_dma_cookie_t *cookie;

                cookie = ddi_dma_coookie_get(handle, i);
                /*
                 * Use the dmac_laddress and dmac_size members to
                 * properly program the device or descriptor rings.
                 */
        }
}
```

SEE ALSO

ddi_dma_addr_bind_handle(9F), ddi_dma_alloc_handle(9F), ddi_dma_buf_bind_handle(9F), ddi_dma_unbind_handle(9F), ddi_dma_cookie(9S)

Writing Device Drivers.