NAME

ddi_soft_state, ddi_get_soft_state, ddi_soft_state_fini, ddi_soft_state_free, ddi_soft_state_init, ddi_soft_state_zalloc – driver soft state utility routines

SYNOPSIS

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

void *ddi_get_soft_state(void **state*, **int** *item***);**

void ddi_soft_state_fini(void ***state_p***);**

void ddi_soft_state_free(void **state*, **int** *item***);**

int ddi_soft_state_init(void ***state_p*, **size_t** *size*, **size_t** *n_items***);**

int ddi_soft_state_zalloc(void **state*, **int** *item***);**

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

ARGUMENTS

state_p	Address of the opaque state pointer which will be initialized by **ddi_soft_state_init( )** to point to implementation dependent data.
size	Size of the item which will be allocated by subsequent calls to **ddi_soft_state_zalloc( )**.
n_items	A hint of the number of items which will be preallocated; zero is allowed.
state	An opaque pointer to implementation-dependent data that describes the soft state.
item	The item number for the state structure; usually the instance number of the associated devinfo node.

DESCRIPTION

Most device drivers maintain state information with each instance of the device they control; for example, a soft copy of a device control register, a mutex that must be held while accessing a piece of hardware, a partition table, or a unit structure. These utility routines are intended to help device drivers manage the space used by the driver to hold such state information.

For example, if the driver holds the state of each instance in a single state structure, these routines can be used to dynamically allocate and deallocate a separate structure for each instance of the driver as the instance is attached and detached.

To use the routines, the driver writer needs to declare a state pointer, *state_p*, which the implementation uses as a place to hang a set of per-driver structures; everything else is managed by these routines.

The routine **ddi_soft_state_init( )** is usually called in the drivers **_init**(9E) routine to initialize the state pointer, set the size of the soft state structure, and to allow the driver to pre-allocate a given number of such structures if required.

The routine **ddi_soft_state_zalloc( )** is usually called in the drivers **attach**(9E) routine. The routine is passed an item number which is used to refer to the structure in subsequent calls to **ddi_get_soft_state( )** and **ddi_soft_state_free( )**. The item number is usually just the instance number of the devinfo node, obtained with **ddi_get_instance**(9F). The routine attempts to allocate space for the new structure, and if the space allocation was successful, **DDI_SUCCESS** is returned to the caller.

A pointer to the space previously allocated for a soft state structure can be obtained by calling **ddi_get_soft_state( )** with the appropriate item number.

The space used by a given soft state structure can be returned to the system using **ddi_soft_state_free( )**. This routine is usually called from the drivers **detach**(9E) entry point.

The space used by all the soft state structures allocated on a given state pointer, together with the housekeeping information used by the implementation can be returned to the system using **ddi_soft_state_fini( )**. This routine can be called from the drivers **_fini**(9E) routine.

The **ddi_soft_state_zalloc( )**, **ddi_soft_state_free( )** and **ddi_get_soft_state( )** routines coordinate access to the underlying data structures in an MT-safe fashion, thus no additional locks should be necessary.

RETURN VALUES

ddi_get_soft_state( ):

NULL The requested state structure was not allocated at the time of the call.

pointer The pointer to the state structure.

ddi_soft_state_init( ):

0 The allocation was successful.

EINVAL Either the *size* parameter was zero, or the *state_p* parameter was invalid.

ddi_soft_state_zalloc( ):

DDI_SUCCESS The allocation was successful.

DDI_FAILURE The routine failed to allocate the storage required; either the *state* parameter was invalid, the item number was negative, or an attempt was made to allocate an item number that was already allocated.

CONTEXT

ddi_soft_state_init( ), and **ddi_soft_state_alloc( )** can be called from user context only, since they may internally call **kmem_zalloc**(9F) with the **KM_SLEEP** flag.

The **ddi_soft_state_fini( )**, **ddi_soft_state_free( )** and **ddi_get_soft_state( )** routines can be called from any driver context.

EXAMPLE

The following example shows how the routines described above can be used in terms of the driver entry points of a character-only driver. The example concentrates on the portions of the code that deal with creating and removing the drivers data structures.

```
typedef struct {
        volatile caddr_t *csr;          /* device registers */
        kmutex_t      csr_mutex;        /* protects 'csr' field */
        unsigned int   state;
        dev_info_t    *dip;             /* back pointer to devinfo */
} devstate_t;
```

```
static void *statep;

int
_init(void)
{
        int error;

        error = ddi_soft_state_init(&statep, sizeof (devstate_t), 0);
        if (error != 0)
                return (error);
        if ((error = mod_install(&modlinkage)) != 0)
                ddi_soft_state_fini(&statep);
        return (error);
}

int
_fini(void)
{
        int error;

        if ((error = mod_remove(&modlinkage)) != 0)
                return (error);
        ddi_soft_state_fini(&statep);
        return (0);
}

static int
xxattach(dev_info_t *dip, ddi_attach_cmd_t cmd)
{
        int instance;
        devstate_t *softc;

        switch (cmd) {
        case DDI_ATTACH:
                instance = ddi_get_instance(dip);
                if (ddi_soft_state_zalloc(statep, instance) != DDI_SUCCESS)
                        return (DDI_FAILURE);
                softc = ddi_get_soft_state(statep, instance);
                softc->dip = dip;
                ...
                return (DDI_SUCCESS);
        default:
                return (DDI_FAILURE);
        }
}
```

```
static int
xxdetach(dev_info_t *dip, ddi_detach_cmd_t cmd)
{
        int instance;

        switch (cmd) {

        case DDI_DETACH:
                instance = ddi_get_instance(dip);
                ...
                ddi_soft_state_free(statep, instance);
                return (DDI_SUCCESS);

        default:
                return (DDI_FAILURE);
        }
}

static int
xxopen(dev_t *devp, int flag, int otyp, cred_t *cred_p)
{
        devstate_t *softc;
        int        instance;

        instance = getminor(*devp);
        if ((softc = ddi_get_soft_state(statep, instance)) == NULL)
                return (ENXIO);
        ...
        softc->state |= XX_IN_USE;
        ...
        return (0);
}
```

SEE ALSO

_fini(9E), **_init**(9E), **attach**(9E), **detach**(9E), **ddi_get_instance**(9F), **getminor**(9F), **kmem_zalloc**(9F)

Writing Device Drivers

WARNINGS

There is no attempt to validate the **item** parameter given to **ddi_soft_state_zalloc( )**; other than it must be a positive signed integer. Therefore very large item numbers may cause the driver to hang forever waiting for virtual memory resources that can never be satisfied.

NOTES

If necessary, a hierarchy of state structures can be constructed by embedding state pointers in higher order state structures.

DIAGNOSTICS

All of the messages described below usually indicate bugs in the driver and should not appear in normal operation of the system.

WARNING: ddi_soft_state_zalloc: bad handle
WARNING: ddi_soft_state_free: bad handle
WARNING: ddi_soft_state_fini: bad handle

The implementation-dependent information kept in the state variable is corrupt.

WARNING: ddi_soft_state_free: null handle
WARNING: ddi_soft_state_fini: null handle

The routine has been passed a null or corrupt state pointer. Check that **ddi_soft_state_init( )** has been called.

WARNING: ddi_soft_state_free: item %d not in range [0..%d]

The routine has been asked to free an item which was never allocated. The message prints out the invalid item number and the acceptable range.