NAME

ddi_prop_update, ddi_prop_update_int_array, ddi_prop_update_int, ddi_prop_update_string_array, ddi_prop_update_string, ddi_prop_update_byte_array – update properties

SYNOPSIS

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

int ddi_prop_update_int_array(dev_t *dev*, **dev_info_t** **dip*, **char** **name*, **int** **data*, **u_int** *nelements***);**

int ddi_prop_update_int(dev_t *dev*, **dev_info_t** **dip*, **char** **name*, **int** *data***);**

int ddi_prop_update_string_array(dev_t *dev*, **dev_info_t** **dip*, **char** **name*, **char** ***data*, **u_int** *nelements***);**

int ddi_prop_update_string(dev_t *dev*, **dev_info_t** **dip*, **char** **name*, **char** **data***);**

int ddi_prop_update_byte_array(dev_t *dev*, **dev_info_t** **dip*, **char** **name*, **u_char** **data*, **u_int** *nelements***);**

ARGUMENTS

dev Device number associated with the device.

dip Pointer to the device info node of device whose property list should be updated.

name String containing the name of the property to be updated.

nelements The number of elements contained in the memory pointed at by *data*.

ddi_prop_update_int_array( )

data A pointer an integer array with which to update the property.

ddi_prop_update_int( )

data An integer value with which to update the property.

ddi_prop_update_string_array( )

data A pointer to a string array with which to update the property. The array of strings is formatted as an array of pointers to NULL terminated strings, much like the *argv* argument to **execve**(2).

ddi_prop_update_string( )

data A pointer to a string value with which to update the property.

ddi_prop_update_byte_array( )

data A pointer to a byte array with which to update the property.

INTERFACE LEVEL

Solaris DDI specific (Solaris DDI).

DESCRIPTION

The property update routines search for and, if found, modify the value of a given property. Properties are searched for based on the *dip*, *name*, *dev*, and the type of the data (integer, string or byte). The driver software properties list is searched. If the property is found, it is updated with the supplied value. If the property is not found on this list, a new property is created with the value supplied. For example, if a driver attempts to update the "foo" property, a property named "foo" is searched for on the driver's software property list. If "foo" is found, the value is updated. If "foo" is not found, a new property named "foo" is created on the driver's software property list with the supplied value even if a "foo" property exists on another property list (such as a PROM property list).

Every property value has a data type associated with it: byte, integer, or string. A property should be updated using a function with the same corresponding data type as the property value. For example, an integer property must be updated using either **ddi_prop_update_int_array( )** or **ddi_prop_update_int( ).** Attempts to update a property with a function that does correspond to the property value data type will result in the creation of another property with the same name. However, the data type of the new property value will correspond to the data type called out in the function name.

Usually, the *dev* argument should be set to the actual device number that this property is associated with. If the property is not associated with any particular *dev*, then the argument *dev* should be set to **DDI_DEV_T_NONE.** This property will then match a look up request (see **ddi_prop_lookup**(9F)) with the *match_dev* argument set to **DDI_DEV_T_ANY.** If no *dev* is available for the device (for example during **attach**(9E) time), one can be created using **makedevice**(9F) with a major number of **DDI_MAJOR_T_UNKNOWN.** The update routines will then generate the correct *dev* when creating or updating the property.

name must always be set to the name of the property being updated.

For the routines **ddi_prop_update_int_array**( ), **ddi_prop_update_string_array**( ), **ddi_prop_update_string**( ), and **ddi_prop_update_byte_array**( ) *data* is a pointer which points to memory containing the value of the property. In each case **data* points to a different type of property value. See the individual descriptions of the routines below for details concerning the different values. *nelements* is an unsigned integer which contains the number of integer, string, or byte elements accounted for in the memory pointed at by **data*.

For the routine **ddi_prop_update_int**( ), *data* is the new value of the property.

ddi_prop_update_int_array( )
Updates or creates an array of integer property values. An array of integers is defined to be *nelements* of 4 byte long integer elements. *data* must be a pointer to an integer array with which to update the property.

ddi_prop_update_int( )
Update or creates a single integer value of a property. *data* must be an integer value with which to update the property.

ddi_prop_update_string_array( )
Updates or creates a property that is an array of strings. *data* must be a pointer to a string array with which to update the property. The array of strings is formatted as an array of pointers to NULL terminated strings, much like the *argv* argument to **execve**(2).

ddi_prop_update_string( )
Updates or creates a property that is a single string value. *data* must be a pointer to a string with which to update the property.

ddi_prop_update_byte_array( ) Updates or creates a property that is an array of bytes. *data* should be a pointer to a byte array with which to update the property.

The property update routines may block to allocate memory needed to hold the value of the property.

RETURN VALUES

All of the property update routines return:

DDI_PROP_SUCCESS on success.

DDI_PROP_INVAL_ARG if an attempt is made to update a property with *name* set to **NULL** or *name* set to the null string.

DDI_PROP_CANNOT_ENCODE If the bytes of the property cannot be encoded.

CONTEXT

These functions can only be called from user or kernel context.

EXAMPLES

The following example demonstrates the use of **ddi_prop_update( )**.

```
int     options[4];

/*
 * Create the "options" integer array with
 * our default values for these parameters
 */
options[0] = XX_OPTIONS0;
options[1] = XX_OPTIONS1;
options[2] = XX_OPTIONS2;
options[3] = XX_OPTIONS3;
i = ddi_prop_update_int_array(xx_dev, xx_dip, "options",
        &options, sizeof (options) / sizeof (int));
```

SEE ALSO

ddi_prop_lookup(9F), **ddi_prop_remove**(9F)

Writing Device Drivers