

```

/* includes */

#include "vxWorks.h"
#include "string.h"
#include "sioLib.h"
#include "errno.h"
#include "ctype.h"

#include "stdio.h"
#include "stdlib.h"
#include "ioLib.h"

#include "usb/usbPlatform.h"
#include "usb/ossLib.h" /* operations system srvc */
#include "usb/usb.h" /* general USB definitions */
#include "usb/usbListLib.h" /* linked list functions */
#include "usb/usbLib.h" /* USB interface */
#include "usb/usbLib.h" /* USB utility functions */
#include "usb/usbHid.h" /* USB HID definitions */
#include "drv/usb/usbButtonsLightsLib.h" /* our API */

/* defines */

#define BTNLTS_CLIENT_NAME "usbButtonsLightsLib" /* our USB client name */

#define BTNLTS_Q_DEPTH 8 /* Max characters in buttonslights queue */

/* If your hardware platform has problems sharing cache lines, then define
 * CACHE_LINE_SIZE below so that critical buffers can be allocated within
 * their own cache lines.
 */

#define CACHE_LINE_SIZE 16

/* typedefs */

/*
 * ATTACH_REQUEST
 */

typedef struct attach_request
{
    LINK reqLink; /* linked list of requests */
    USB_BTNLTS_ATTACH_CALLBACK callback; /* client callback routine */
    pVOID callbackArg; /* client callback argument */
} ATTACH_REQUEST, *pATTACH_REQUEST;

/* USB_BTNLTS_SIO_CHAN is the internal data structure we use to track each USB
 * buttonslights.
 */

typedef struct usb_btnlts_sio_chan
{
    SIO_CHAN sioChan; /* must be first field */

    LINK sioLink; /* linked list of buttonslights structs */

    UINT16 lockCount; /* Count of times structure locked */

    USBD_NODE_ID nodeId; /* buttonslights node Id */
    UINT16 configuration; /* configuration/interface reported as */
    UINT16 interface; /* a buttonslights by this device */

    BOOL connected; /* TRUE if buttonslights currently connected */

    USBD_PIPE_HANDLE pipeHandle; /* USBD pipe handle for interrupt pipe */
    USB_IRP irp; /* IRP to monitor interrupt pipe */
    BOOL irpInUse; /* TRUE while IRP is outstanding */

    char btn_statusReq;

    int mode; /* SIO_MODE_INT or SIO_MODE_POLL */

```

```

    } USB_BTNLTS_SIO_CHAN, *pUSB_BTNLTS_SIO_CHAN;

/* forward static declarations */
LOCAL VOID usbButtonsLightsIrpCallback (pVOID p);

/* locals */

LOCAL UINT16 initCount = 0; /* Count of init nesting */

LOCAL MUTEX_HANDLE btnltsMutex; /* mutex used to protect internal structs */

LOCAL LIST_HEAD sioList; /* linked list of USB_BTNLTS_SIO_CHAN */
LOCAL LIST_HEAD reqList; /* Attach callback request list */

LOCAL USBD_CLIENT_HANDLE usbdHandle; /* our USBD client handle */

/*****
 *
 * initBtnltsIrp - Initialize IRP to listen for input on interrupt pipe
 *
 * RETURNS: TRUE if able to submit IRP successfully, else FALSE
 */
LOCAL BOOL initBtnltsIrp
(
    pUSB_BTNLTS_SIO_CHAN pSioChan
)
{
    pUSB_IRP pIrp = &pSioChan->irp;

    /* Initialize IRP */

    memset (pIrp, 0, sizeof (*pIrp));

    pIrp->userPtr = pSioChan;
    pIrp->irpLen = sizeof (*pIrp);
    pIrp->userCallback = usbButtonsLightsIrpCallback;
    pIrp->timeout = USB_TIMEOUT_NONE;
    pIrp->transferLen = 1;

    pIrp->bfrCount = 1;
    pIrp->bfrList [0].pid = USB_PID_IN;
    pIrp->bfrList [0].pBfr = &(pSioChan->btn_statusReq);
    pIrp->bfrList [0].bfrLen = 1;

    /* Submit IRP */

    if (usbdTransfer (usbdHandle, pSioChan->pipeHandle, pIrp) != OK)
        return FALSE;

    pSioChan->irpInUse = TRUE;

    return TRUE;
}

/*****
 *
 * usbButtonsLightsIrpCallback - Invoked upon IRP completion/cancellation
 *
 * Examines the cause of the IRP completion. If completion was successful,
 * interprets the USB buttonslights's boot report and re-submits the IRP.
 *
 * RETURNS: N/A
 */
LOCAL VOID usbButtonsLightsIrpCallback
(
    pVOID p /* completed IRP */
)

```

```

{
static UINT8 ledReport = 0xff;
UINT8 pre_ledStatus;

pUSB_IRP pIrp = (pUSB_IRP) p;
pUSB_BTNLTS_SIO_CHAN pSioChan = pIrp->userPtr;

OSS_MUTEX_TAKE (btnltsMutex, OSS_BLOCK);

/* Was the IRP successful? */

if (pIrp->result == OK)
{
/* Interpret the buttonslights report */

pre_ledStatus = ledReport;

ledReport = pIrp->bfrList[0].pBfr[0];

if(pre_ledStatus != ledReport)
printf(" LED Status : %d \n", ledReport);

usbHidReportSet (usbHandle, pSioChan->nodeId, pSioChan->interface,
USB_HID_RPT_TYPE_OUTPUT, 0, &ledReport, sizeof (ledReport));

}

/* Re-submit the IRP unless it was canceled - which would happen only
* during pipe shutdown (e.g., the disappearance of the device).
*/

pSioChan->irpInUse = FALSE;

if (pIrp->result != S_usbHcdLib_IRP_CANCELED)
initBtnltsIrp (pSioChan);

OSS_MUTEX_RELEASE (btnltsMutex);
}

/*****
*
* configureSioChan - configure USB buttonslights for operation
*
* Selects the configuration/interface specified in the <pSioChan>
* structure. These values come from the USB_D dynamic attach callback,
* which in turn retrieved them from the configuration/interface
* descriptors which reported the device to be a buttonslights.
*
* RETURNS: TRUE if successful, else FALSE if failed to configure channel
*/
LOCAL BOOL configureSioChan
(
pUSB_BTNLTS_SIO_CHAN pSioChan
)
{
pUSB_CONFIG_DESCR pCfgDescr;
pUSB_INTERFACE_DESCR pIfDescr;
pUSB_ENDPOINT_DESCR pEpDescr;
UINT8 bfr [USB_MAX_DESCR_LEN];
pUINT8 pBfr;
UINT16 actLen;
UINT16 ifNo;
UINT16 maxPacketSize;

/* Read the configuration descriptor to get the configuration selection
* value and to determine the device's power requirements.
*/

if (usbDescriptorGet (usbHandle, pSioChan->nodeId,
USB_RT_STANDARD | USB_RT_DEVICE, USB_DESCR_CONFIGURATION, 0, 0,
sizeof (bfr), bfr, &actLen) != OK)

```

```

return FALSE;

if ((pCfgDescr = usbDescrParse (bfr, actLen, USB_DESCR_CONFIGURATION))
== NULL)
return FALSE;

/* Look for the interface indicated in the pSioChan structure. */

ifNo = 0;
pBfr = bfr;

while ((pIfDescr = usbDescrParseSkip (&pBfr, &actLen, USB_DESCR_INTERFACE))
!= NULL)
{
if (ifNo == pSioChan->interface)
break;
ifNo++;
}

if (pIfDescr == NULL)
return FALSE;

/* Retrieve the endpoint descriptor following the identified interface
* descriptor.
*/

if ((pEpDescr = usbDescrParseSkip (&pBfr, &actLen, USB_DESCR_ENDPOINT))
== NULL)
return FALSE;

/* Select the configuration. */

if (usbConfigurationSet (usbHandle, pSioChan->nodeId,
pCfgDescr->configurationValue,
pCfgDescr->maxPower * USB_POWER_MA_PER_UNIT) != OK)
return FALSE;

/* Select interface
*
* NOTE: Some devices may reject this command, and this does not represent
* a fatal error. Therefore, we ignore the return status.
*/

usbInterfaceSet (usbHandle, pSioChan->nodeId,
pSioChan->interface, pIfDescr->alternateSetting);

/* Create a pipe to monitor input reports from the buttons/leds. */

maxPacketSize = *((pUINT8) &pEpDescr->maxPacketSize) |
*((pUINT8) &pEpDescr->maxPacketSize) + 1) << 8);

if (usbPipeCreate (usbHandle, pSioChan->nodeId,
pEpDescr->endpointAddress, pCfgDescr->configurationValue,
pSioChan->interface, USB_XFRTYPE_INTERRUPT, USB_DIR_IN,
maxPacketSize, 100,
pEpDescr->interval, &pSioChan->pipeHandle) != OK)
return FALSE;

/* Initiate IRP to listen for input on interrupt pipe */
if (!initBtnLedsIrp (pSioChan))
return FALSE;

return TRUE;
}

/*****
*
* destroyAttachRequest - disposes of an ATTACH_REQUEST structure
*
* RETURNS: N/A
*/
LOCAL VOID destroyAttachRequest
(

```

```

    pATTACH_REQUEST pRequest
    )

    {
    /* Unlink request */

    usbListUnlink (&pRequest->reqLink);

    /* Dispose of structure */

    OSS_FREE (pRequest);
    }

/*****
*
* destroySioChan - disposes of a USB_BTNLTS_SIO_CHAN structure
*
* Unlinks the indicated USB_BTNLTS_SIO_CHAN structure and de-allocates
* resources associated with the channel.
*
* RETURNS: N/A
*/

LOCAL VOID destroySioChan
(
    pUSB_BTNLTS_SIO_CHAN pSioChan
)

{
    /* Unlink the structure. */

    usbListUnlink (&pSioChan->sioLink);

    /* Release pipe if one has been allocated.  Wait for the IRP to be
    * cancelled if necessary.
    */

    if (pSioChan->pipeHandle != NULL)
        usbdPipeDestroy (usbdHandle, pSioChan->pipeHandle);

    while (pSioChan->irpInUse)
        OSS_THREAD_SLEEP (1);

    /* Release structure. */

    OSS_FREE (pSioChan);
}

/*****
*
* createSioChan - creates a new USB_BTNLTS_SIO_CHAN structure
*
* Creates a new USB_BTNLTS_SIO_CHAN structure for the indicated <nodeId>.
* If successful, the new structure is linked into the sioList upon
* return.
*
* <configuration> and <interface> identify the configuration/interface
* that first reported itself as a buttonsights for this device.
*
* RETURNS: pointer to newly created structure, or NULL if failure
*/

LOCAL pUSB_BTNLTS_SIO_CHAN createSioChan
(
    USBD_NODE_ID nodeId,
    UUINT16 configuration,
    UUINT16 interface
)

{
    pUSB_BTNLTS_SIO_CHAN pSioChan;
    UUINT16 i;

    /* Try to allocate space for a new buttonsights struct */
    if ((pSioChan = OSS_CALLOC (sizeof (*pSioChan))) == NULL)

```

```

return NULL;

pSioChan->nodeId = nodeId;
pSioChan->connected = TRUE;
pSioChan->mode = SIO_MODE_POLL;

pSioChan->configuration = configuration;
pSioChan->interface = interface;

pSioChan->btn_statusReq = 0x31;

/* Try to configure the buttonslights. */
if (!configureSioChan (pSioChan))
{
destroySioChan (pSioChan);
return NULL;
}

/* Link the newly created structure. */

usbListLink (&sioList, pSioChan, &pSioChan->sioLink, LINK_TAIL);

return pSioChan;
}

/*****
*
* findSioChan - Searches for a USB_BTNLTS_SIO_CHAN for indicated node ID
*
* RETURNS: pointer to matching USB_BTNLTS_SIO_CHAN or NULL if not found
*/

LOCAL pUSB_BTNLTS_SIO_CHAN findSioChan
(
    USBD_NODE_ID nodeId
)
{
    pUSB_BTNLTS_SIO_CHAN pSioChan = usbListFirst (&sioList);

    while (pSioChan != NULL)
    {
        if (pSioChan->nodeId == nodeId)
            break;

        pSioChan = usbListNext (&pSioChan->sioLink);
    }

    return pSioChan;
}

/*****
*
* notifyAttach - Notifies registered callers of attachment/removal
*
* RETURNS: N/A
*/

LOCAL VOID notifyAttach
(
    pUSB_BTNLTS_SIO_CHAN pSioChan,
    UINT16 attachCode
)
{
    pATTACH_REQUEST pRequest = usbListFirst (&reqList);

    while (pRequest != NULL)
    {
        (*pRequest->callback) (pRequest->callbackArg,
            (SIO_CHAN *) pSioChan, attachCode);

        pRequest = usbListNext (&pRequest->reqLink);
    }
}

```

```

/*****
*
* usbButtonsLightsAttachCallback - called by USBBD when buttonslights
* attached/removed
*
* RETURNS: N/A
*/

LOCAL VOID usbButtonsLightsAttachCallback
(
    USBBD_NODE_ID nodeId,
    UINT16 attachAction,
    UINT16 configuration,
    UINT16 interface,
    UINT16 deviceClass,
    UINT16 deviceSubClass,
    UINT16 deviceProtocol
)
{
    pUSB_BTNLTS_SIO_CHAN pSioChan;

    OSS_MUTEX_TAKE (btnltsMutex, OSS_BLOCK);

    /* Depending on the attach code, add a new buttonslights or disabled one
    * that's already been created.
    */

    switch (attachAction)
    {
    case USBBD_DYNA_ATTACH:

        /* A new device is being attached. Check if we already
        * have a structure for this device.
        */

        printf("usbButtonsLightsAttachCallback : USBBD Dynamic Attach \n");
        if (findSioChan (nodeId) != NULL)
            break;

        /* Create a new structure to manage this device. If there's
        * an error, there's nothing we can do about it, so skip the
        * device and return immediately.
        */

        if ((pSioChan = createSioChan (nodeId, configuration, interface))
            == NULL)
            break;

        /* Notify registered callers that a new buttonslights has been
        * added and a new channel created.
        */

        notifyAttach (pSioChan, USB_BTNLTS_ATTACH);

        break;

    case USBBD_DYNA_REMOVE:

        /* A device is being detached. Check if we have any
        * structures to manage this device.
        */

        printf("usbButtonsLightsAttachCallback : USBBD Dynamic Remove \n");

        if ((pSioChan = findSioChan (nodeId)) == NULL)
            break;

        /* The device has been disconnected. */

        pSioChan->connected = FALSE;

        /* Notify registered callers that the buttonslights has been
        * removed and the channel disabled.
        *
        * NOTE: We temporarily increment the channel's lock count
        * to prevent usbButtonsLightsSioChanUnlock() from destroying the

```

```

    * structure while we're still using it.
    */

pSioChan->lockCount++;

notifyAttach (pSioChan, USB_BTNLTS_REMOVE);

pSioChan->lockCount--;

/* If no callers have the channel structure locked, destroy
 * it now. If it is locked, it will be destroyed later during
 * a call to usbButtonsLightsUnlock().
 */

if (pSioChan->lockCount == 0)
destroySioChan (pSioChan);

break;
}

OSS_MUTEX_RELEASE (btnltsMutex);
}

/*****
 *
 * doShutdown - shuts down USB buttonslights SIO driver
 *
 * <errCode> should be OK or S_usbButtonsLightsLib_xxxx. This value will be
 * passed to ossStatus() and the return value from ossStatus() is the
 * return value of this function.
 *
 * RETURNS: OK, or ERROR per value of <errCode> passed by caller
 */

LOCAL STATUS doShutdown
(
    int errCode
)
{
    pATTACH_REQUEST pRequest;
    pUSB_BTNLTS_SIO_CHAN pSioChan;

    /* Dispose of any outstanding notification requests */

    while ((pRequest = usbListFirst (&reqList)) != NULL)
destroyAttachRequest (pRequest);

    /* Dispose of any open buttonslights connections. */

    while ((pSioChan = usbListFirst (&sioList)) != NULL)
destroySioChan (pSioChan);

    /* Release our connection to the USB. The USB automatically
     * releases any outstanding dynamic attach requests when a client
     * unregisters.
     */

    if (usbHandle != NULL)
    {
        usbClientUnregister (usbHandle);
        usbHandle = NULL;
    }

    /* Release resources. */

    if (btnltsMutex != NULL)
    {
        OSS_MUTEX_DESTROY (btnltsMutex);
        btnltsMutex = NULL;
    }

    return ossStatus (errCode);
}

```

```

}

/*****
*
* usbButtonsLightsDevInit - initialize USB buttonslights SIO driver
*
* Initializes the USB buttonslights SIO driver. The USB buttonslights SIO driver
* maintains an initialization count, so calls to this function may be
* nested.
*
* RETURNS: OK, or ERROR if unable to initialize.
*
* ERRNO:
*   S_usbButtonsLightsLib_OUT_OF_RESOURCES
*   S_usbButtonsLightsLib_USBD_FAULT
*/

STATUS usbButtonsLightsDevInit (void)
{
    /* If not already initialized, then initialize internal structures
    * and connection to USBD.
    */

    if (initCount == 0)
    {
        /* Initialize lists, structures, resources. */

        memset (&sioList, 0, sizeof (sioList));
        memset (&reqList, 0, sizeof (reqList));
        btnltsMutex = NULL;
        usbdHandle = NULL;

        if (OSS_MUTEX_CREATE (&btnltsMutex) != OK)
            return doShutdown (S_usbButtonsLightsLib_OUT_OF_RESOURCES);

        /* Establish connection to USBD */

        if (usbdClientRegister (BTNLTS_CLIENT_NAME, &usbdHandle) != OK ||
            usbdDynamicAttachRegister (usbdHandle, USB_CLASS_HID,
            USB_SUBCLASS_HID_NONE, USB_PROTOCOL_HID_BOOT_NONE,
            usbButtonsLightsAttachCallback) != OK)
        {
            return doShutdown (S_usbButtonsLightsLib_USBD_FAULT);
        }
    }

    initCount++;

    return OK;
}

/*****
*
* usbButtonsLightsDevShutdown - shuts down buttonslights SIO driver
*
* RETURNS: OK, or ERROR if unable to shutdown.
*
* ERRNO:
*   S_usbButtonsLightsLib_NOT_INITIALIZED
*/

STATUS usbButtonsLightsDevShutdown (void)
{
    /* Shut down the USB buttonslights SIO driver if the initCount goes to 0. */

    if (initCount == 0)
        return ossStatus (S_usbButtonsLightsLib_NOT_INITIALIZED);

    if (--initCount == 0)
        return doShutdown (OK);

    return OK;
}

```

```

/*****
*
* usbButtonsLightsDynamicAttachRegister - Register buttonslights attach callback
*
* <callback> is a caller-supplied function of the form:
*
* .CS
* typedef (*USB_BTNLTS_ATTACH_CALLBACK)
* (
*     pVOID arg,
*     SIO_CHAN *pSioChan,
*     UINT16 attachCode
* );
* .CE
*
* usbButtonsLightsLib will invoke <callback> each time a USB buttonslights
* is attached to or removed from the system. <arg> is a caller-defined
* parameter which will be passed to the <callback> each time it is
* invoked. The <callback> will also be passed a pointer to the
* SIO_CHAN structure for the channel being created/destroyed and
* an attach code of USB_BTNLTS_ATTACH or USB_BTNLTS_REMOVE.
*
* RETURNS: OK, or ERROR if unable to register callback
*
* ERRNO:
*     S_usbButtonsLightsLib_BAD_PARAM
*     S_usbButtonsLightsLib_OUT_OF_MEMORY
*/

STATUS usbButtonsLightsDynamicAttachRegister
(
    USB_BTNLTS_ATTACH_CALLBACK callback, /* new callback to be registered */
    pVOID arg /* user-defined arg to callback */
)
{
    pATTACH_REQUEST pRequest;
    pUSB_BTNLTS_SIO_CHAN pSioChan;
    int status = OK;

    /* Validate parameters */

    if (callback == NULL)
        return ossStatus (S_usbButtonsLightsLib_BAD_PARAM);

    OSS_MUTEX_TAKE (btnltsMutex, OSS_BLOCK);

    /* Create a new request structure to track this callback request. */

    if ((pRequest = OSS_CALLOC (sizeof (*pRequest))) == NULL)
        status = S_usbButtonsLightsLib_OUT_OF_MEMORY;
    else
    {
        pRequest->callback = callback;
        pRequest->callbackArg = arg;

        usbListLink (&reqList, pRequest, &pRequest->reqLink, LINK_TAIL);

        /* Perform an initial notification of all currently attached
         * buttonslights devices.
         */

        pSioChan = usbListFirst (&sioList);

        while (pSioChan != NULL)
        {
            if (pSioChan->connected)
                (*callback) (arg, (SIO_CHAN *) pSioChan, USB_BTNLTS_ATTACH);

            pSioChan = usbListNext (&pSioChan->sioLink);
        }
    }

    OSS_MUTEX_RELEASE (btnltsMutex);
}

```

```

    return ossStatus (status);
}

/*****
*
* usbButtonsLightsDynamicAttachUnregister - Unregisters buttonslights attach callback
*
* This function cancels a previous request to be dynamically notified for
* buttonslights attachment and removal. The <callback> and <arg> paramters must
* exactly match those passed in a previous call to
* usbButtonsLightsDynamicAttachRegister().
*
* RETURNS: OK, or ERROR if unable to unregister callback
*
* ERRNO:
*   S_usbButtonsLightsLib_NOT_REGISTERED
*/

STATUS usbButtonsLightsDynamicAttachUnRegister
(
    USB_BTNLTS_ATTACH_CALLBACK callback, /* callback to be unregistered */
    PVOID arg /* user-defined arg to callback */
)
{
    pATTACH_REQUEST pRequest;
    int status = S_usbButtonsLightsLib_NOT_REGISTERED;

    OSS_MUTEX_TAKE (btnltsMutex, OSS_BLOCK);

    pRequest = usbListFirst (&reqList);

    while (pRequest != NULL)
    {
        if (callback == pRequest->callback && arg == pRequest->callbackArg)
        {
            /* We found a matching notification request. */

            destroyAttachRequest (pRequest);
            status = OK;
            break;
        }

        pRequest = usbListNext (&pRequest->reqLink);
    }

    OSS_MUTEX_RELEASE (btnltsMutex);

    return ossStatus (status);
}

/*****
*
* usbButtonsLightsSioChanLock - Marks SIO_CHAN structure as in use
*
* A caller uses usbButtonsLightsSioChanLock() to notify usbButtonsLightsLib that
* it is using the indicated SIO_CHAN structure. usbButtonsLightsLib maintains
* a count of callers using a particular SIO_CHAN structure so that it
* knows when it is safe to dispose of a structure when the underlying
* USB buttonslights is removed from the system. So long as the "lock count"
* is greater than zero, usbButtonsLightsLib will not dispose of an SIO_CHAN
* structure.
*
* RETURNS: OK, or ERROR if unable to mark SIO_CHAN structure in use.
*/

STATUS usbButtonsLightsSioChanLock
(
    SIO_CHAN *pChan /* SIO_CHAN to be marked as in use */
)
{
    {
        pUSB_BTNLTS_SIO_CHAN pSioChan = (pUSB_BTNLTS_SIO_CHAN) pChan;
        pSioChan->lockCount++;

        return OK;
    }
}

```

```

}

/*****
*
* usbButtonsLightsSioChanUnlock - Marks SIO_CHAN structure as unused
*
* This function releases a lock placed on an SIO_CHAN structure.  When a
* caller no longer needs an SIO_CHAN structure for which it has previously
* called usbButtonsLightsSioChanLock(), then it should call this function to
* release the lock.
*
* NOTE: If the underlying USB buttonslights device has already been removed
* from the system, then this function will automatically dispose of the
* SIO_CHAN structure if this call removes the last lock on the structure.
* Therefore, a caller must not reference the SIO_CHAN again structure after
* making this call.
*
* RETURNS: OK, or ERROR if unable to mark SIO_CHAN structure unused
*
* ERRNO:
*   S_usbButtonsLightsLib_NOT_LOCKED
*/

STATUS usbButtonsLightsSioChanUnlock
(
    SIO_CHAN *pChan    /* SIO_CHAN to be marked as unused */
)
{
    pUSB_BTNLTS_SIO_CHAN pSioChan = (pUSB_BTNLTS_SIO_CHAN) pChan;
    int status = OK;

    OSS_MUTEX_TAKE (btnltsMutex, OSS_BLOCK);

    if (pSioChan->lockCount == 0)
        status = S_usbButtonsLightsLib_NOT_LOCKED;
    else
    {
        /* If this is the last lock and the underlying USB buttonslights is
        * no longer connected, then dispose of the buttonslights.
        */

        if (--pSioChan->lockCount == 0 && !pSioChan->connected)
            destroySioChan (pSioChan);
    }

    OSS_MUTEX_RELEASE (btnltsMutex);

    return ossStatus (status);
}

/* end of file. */

```