E

LSL Support Procedures

Introduction

MLIDs that are developed using the MSM do not need to make any calls to Link Support Layer routines. Because sample code provided shows calls to the LSL, this section is provided as a reference for developers. Novell recommends that developers not use these calls, but use the standard MSM interface. The developer may use the routines in this chapter if some portion of the driver absolutely must interact directly with the LSL.

Table E.1 is a list of the completion codes returned by the LSL.

Table E.1 LSL Completion Codes

Completion Code	Message
00000000h	Successful
FFFFFF81h	BadCommand
FFFFFF82h	BadParameters
FFFFFF83h	DuplicateEntry
FFFFFF84h	Fail
FFFFFF85h	ItemNotPresent
FFFFFF86h	NoMoreItems
FFFFFF87h	NoSuchDriver
FFFFFF88h	NoSuchHandles
FFFFFF89h	OutOfResources
FFFFFF8Ah	RxOverflow
FFFFFF8Bh	InCriticalSection
FFFFFF8Ch	TransmitFailed
FFFFFF8Dh	PacketUndeliverable
FFFFFFFCh	Cancelled

LSLAddProtocolID

On Entry

EAX	points to the 6 byte protocol ID (PID) being added
ECX	points to the frame type string (which is byte-length preceded and zero-terminated)
EDX	points to the protocol stack ID string (which is byte-length preceded and zero-terminated)
Interrupts	are in any state
Call	at process time only

On Return

EAX	has a completion code (Normally drivers should ignore the completion code.)
Flags	Zero flag is set according to EAX
Interrupts	are preserved
Note	all other registers are destroyed

Completion Codes

00000000h	Successful: The LSL successfully added the new Protocol ID.
FFFFFF82h	BadParameters: The length of the string parameter exceeded 15 characters.
FFFFFF83h	DuplicateEntry: There is already a protocol ID registered for the given media/stack combination.
FFFFFF89h	OutOfResources: The LSL had no resource to register another Protocol ID.

Description

LSLAddProtocolID allows the driver to tell the LSL the names and protocol ID (PID) of each protocol stack it supports.

The driver's initialization procedure should call this routine to add the default PID for IPX.

IPX PIDs for some frame types are:

Ethernet_802.3	0
Ethernet_II	8137h
Ethernet_802.2	E0h
Ethernet_Snap	8137h
Token-Ring	E0h
Token-Ring_Snap	8137h

Note: If the PID values are less than 6 bytes, pad the most significant bytes of the 6-byte PID with zeroes.

Example

```
ProtocolID        db 0, 0, 0, 0, 81h, 37h ;EII PID
ProtocolName      db 3, 'IPX', 0          ;IPX Protocol Stack
FrameTypeString   db 11, 'ETHERNET_II', 0 ;EII String

lea   eax, ProtocolID                     ;Pointer to 6-byte PID
lea   edx, ProtocolName                   ;Pointer to protocol name string
lea   ecx, FrameTypeString                ;Pointer to frame type string

call  LSLAddProtocolID
jmp   DoneAddingProtocolTypes
```

LSLDeRegisterMLID

On Entry

EBX	has the board number
Interrupts	are in any state
Call	at process time only

On Return

EAX	has a completion code
Interrupts	are disabled, but could have been enabled
Note	all other registers are destroyed

Completion Codes

00000000h	Successful: The LSL successfully deregistered the MLID.
FFFFFF82h	BadParameters: The LSL did not have an MLID registered as the board number passed in EBX.

Description

The driver calls *LSLDeRegisterMLID* to deregister a logical board from the LSL and to inform all protocol stacks bound to that board that the board is no longer available.

If the adapter is not having trouble sending out packets, the driver should use *LSLUnBindThenDeRegisterMLID*.

Example

```
push  ebp                            ;DeRegister destroys all registers
push  ebx
movzx ebx, [ebx].CDriverBoardNumber ;Get the board number
call  LSLDeRegisterMLID              ;DeRegisterMLID
pop   ebx
pop   ebp
```

LSLFastRcvEvent

On Entry

ESI	points to the receive buffer to be processed
Interrupts	are in any state
Call	at process or interrupt time

On Return

Interrupts	are disabled, but could have been enabled
Note	all registers are destroyed

Description

This routine improves the performance of drivers that call *LSLServiceEvents* immediately after calling *LSLHoldRcvEvent*. *LSLFastRcvEvent* dispatches the ECB directly to the protocol stack.

Be aware that *LSLFastRcvEvent* may enable interrupts. Consequently, if the board service routine runs with interrupts disabled, you may want to structure the driver so that either this is the last call the board service routine makes before issuing a ret, or that the board service routine can handle being re-entered at the point where *LSLFastRcvEvent* is called.

If the board service routine masks off the PIC instead of disabling interrupts, you can use *LSLFastRcvEvent* at any point in the receive routine without worrying about being re-entered.

The driver must ensure that the following fields of the ECB are filled in before calling this routine:

ProtocolID
BoardNumber
ImmediateAddress
DriverWorkspace
PacketLength
FragmentOffset
FragmentSize

Note: This process may call the *DriverSend* routine of the calling board and may enable the interrupts.

Example

```
mov   esi, ECBHoldBuffer
call  LSLFastRcvEvent
```

LSLFastSendComplete

On Entry

ESI	points to the ECB that was sent
Interrupts	are in any state
Call	at process or interrupt time

On Return

Interrupts	are disabled, but could have been enabled
Note	all registers are destroyed

Description

This routine improves the performance of drivers that call *LSLServiceEvents* immediately after calling *LSLSendComplete*. *LSLFastSendComplete* immediately returns the ECB to the LSL.

Be aware that *LSLFastSendComplete* may enable interrupts. Consequently, the send routine could be re-entered before *LSLFastSendComplete* returns.

Example

```
push  ebp                  ;Save pointer to adapter data space
call  LSLFastSendComplete  ;Clean up ECB
pop   ebp                  ;Restore pointer to adapter data space
```

LSLGetMaximumPacketSize

On Entry

Interrupts	are in any state
Call	at process or interrupt time

On Return

EAX	has the maximum physical packet size that the LSL supports.
Interrupts	are preserved
Note	all other registers are preserved

Description

LSLGetMaximumPacketSize returns the maximum packet size the LSL can accommodate.

Example

```
call  LSLGetMaximumPacketSize ;EAX contains maximum packet size
```

LSLGetSizedRcvECBRTag

On Entry

EAX	points to valid resource tag
ESI	contains the packet size, including all headers
Interrupts	are in any state
Call	at process or interrupt time

On Return

EAX	has a completion code
ESI	points to the receive ECB
Z flag	set according to EAX
Interrupts	are disabled
Note	no other registers are destroyed

Completion Codes

00000000h	Successful: No errors occurred.
FFFFFF89h	OutofResources: The packet size exceeded the maximum ECB size or an ECB was not available.

Description

The driver calls *LSLGetSizedRcvECBRTag* to get a receive buffer for a received packet. The LSL returns an ECB with a buffer large enough to hold the received frame. The length passed in the ESI register should contain the length of all protocol and hardware headers. For example, for an Ethernet II frame, pass DataLength + 14. If a receive ECB is not available, discard the packet.

Drivers that take advantage of bus-mastering DMA must pre-allocate ECBs. These drivers should make a call to *LSLGetMaximumPacketSize* and then put either the returned value **or** the maximum packet length the board can receive--whichever is less--into ESI before calling *LSLGetSizedRcvECBRTag*.

Example

```
mov   esi, ReceiveHeaderRByteCount  ;Get packet size from card
mov   eax, ECBRTag                  ;Get resource tag
call  LSLGetSizedRcvECBRTag         ;Get ECB
jnz   NoECBAvailable                ;Keep copy in ECX
```

LSLHoldRcvEvent

On Entry

ESI	points to the receive ECB to be processed
Interrupts	are in any state
Call	at process or interrupt time

On Return

ESI	preserved
EDI	preserved
EBP	preserved
Interrupts	are disabled, and the call does not enable interrupts.

Description

If the driver does not use *LSLFastRcvEvent*, *LSLHoldRcvEvent* may be called to hand a receive ECB (together with a received packet) to the LSL.

The following fields should be set prior to calling this routine:

ProtocolID
BoardNumber
ImmediateAddress
DriverWorkSpace (Most Significant Byte with destination address type)
packetlength
PacketOffset
PacketSize

Note: The driver cannot modify any fields in the ECB after making this call.

After calling *LSLGetSizedRcvECBRTag* and reading the packet into the receive ECB, the board service routine calls *LSLHoldRcvEvent* to queue the receive ECB on the LSL's hold queue. Before leaving the board service routine, the driver calls *LSLServiceEvents* to dispatch the ECBs on the hold queue.

Example

```
call  LSLHoldRcvEvent   ;ESI points to the ECB
```

LSLRegisterMLIDRTag

On Entry

EAX	points to the MLID send routine
EBX	contains the MLID resource tag
ECX	points to the MLID configuration table
EDX	contains the Loadable Module Handle (this is passed to the driver at initialization. See Figure 6.1)
ESI	points to the driver control handler routine
Interrupts	are in any state
Call	only at process time

On Return

EAX	has a completion code
EBX	has the assigned board number
ECX	has the maximum buffer size of receive ECBs
Z flag	set according to EAX
Interrupts	are preserved
Note	all other registers are destroyed

Completion Codes

000000000h	Successful: No errors occurred.
FFFFFF89h	OutofResources: There was not enough memory to register MLID.
FFFFFF82h	BadParameters: The resource tag was invalid.

Description

The driver's initialization procedure calls *LSLRegisterMLIDRTag* to register a logical board.

By making this call, *DriverInitialize* gives the LSL pointers to a send procedure, a control procedure, and the configuration table for the logical board.

The driver should adjust the three packet size fields--*CDriverMaximumSize*, *CDriverMaxRecvSize*, *CDriverRecvSize*--according to the "Maximum Packet Size Table" shown in Chapter 4.

Example

```
mov   ecx, ebx                      ;ECX points to configuration table
mov   ebx, MLIDRTag                 ;EBX points to MLID resource tag
mov   eax, OFFSET DriverSend        ;EAX points to MLID send routine
mov   esi, OFFSET DriverControl     ;ESI points to the driver control
                                    ;routine
mov   edx, [ESP + Parm0]            ;EDX has the loaded module handle
push  ecx                           ;Save BoardBase
push  ebp                           ;Save AdapterBase

call  LSLRegisterMLIDRTag           ;Register MLID

pop   ebp                           ;Restore AdapterBase
pop   edx                           ;Restore BoardBase into EDX
jnz   ErrorRegisteringMLID          ;Exit initialization if error
                                    ;registering

mov   [edx].CDriverBoardNumber, bx

cmp   [edx].CDriverMaximumSize, ecx ;Adjust packet size fields?
jbe   Short NoAdjust

mov   [edx].CDriverMaximumSize, ecx
sub   ecx, FrameHeaderSize
mov   [edx].CDriverMaxRecvSize, ecx
mov   [edx].CDriverRecvSize, ecx

NoAdjust:
```

LSLReturnRcvECB

On Entry

ESI	points to the receive ECB
Interrupts	are in any state
Call	at process or interrupt time

On Return

EAX	destroyed
Interrupts	are disabled
Note	All other registers preserved

Description

The driver calls *LSLReturnRcvECB* to return an unneeded receive ECB to the LSL.

Example

```
call  LSLReturnRcvECB   ;Return ECB
```

LSLSendComplete

On Entry

ESI	points to the ECB that was sent
Interrupts	are in any state
Call	at process or interrupt time

On Return

EAX	destroyed
Interrupts	are disabled, and will not have been enabled

Description

If the driver does not use *LSLFastSendComplete*, it calls *LSLSendComplete* to return a send ECB to the LSL after it has finished processing the ECB. This call does not return the ECB to its owner; it simply queues the ECB and returns. The driver should call *LSLServiceEvents* at the end of the board service routine and/or *DriverSend* procedure.

Example

```
call  GetNextSend       ;Anything in send queue?
jnz   PollAgain         ;No:  Check for receives

call  StartSend         ;Yes:  Initiate a send
call  LSLSendComplete   ;Queue ECB
jmp   PollAgain         ;Check for receives
```

LSLServiceEvents

On Entry

Interrupts	are in any state
Call	at process or interrupt time

On Return

Interrupts	are disabled, but could have been enabled
Note	all registers are destroyed

Description

If the driver does not use *LSLFastRcvEvent* or *LSLFastSendComplete*, it must call *LSLServiceEvents* to unqueue any packets that were queued by *LSLHoldRcvEvent* or *LSLSendComplete*.

The board service routine calls *LSLServiceEvents* after processing all sends or receives. This is the last thing the board service routine does before returning. All hardware processing must be completed, and the board service routine must be ready to be called by a new interrupt.

The *LSLServiceEvents* routine routes all receive packets to the correct protocol stack.

Note: If the driver uses *LSLFastSendComplete* and *LSLFastHoldRcvEvent* for completing events, it does not need to call *LSLServiceEvents*.

Example

```
call  LSLServiceEvents  ;Let OS service queue
ret
```

LSLUnBindThenDeRegisterMLID

On Entry

EBX	has the board number
Interrupts	are in any state
Call	only at process time
Note	LAN board must not be in a critical section

On Return

Interrupts	are disabled, but could have been enabled
Note	all other registers are destroyed

Description

The driver's *DriverRemove* procedure calls this procedure to unbind the specified LAN board from all protocol stacks and then deregister the board. The driver's remove procedure should call this routine (or *LSLDeRegisterMLID*) for each logical board that the physical card supports.

This routine is identical to *LSLDeRegisterMLID* with the addition that *LSLUnbindThenDeRegisterMLID* allows protocol stacks to attempt to transmit packets advising other machines on the network that this connection is going down. For this reason, you should not use this call in situations where the hardware is having trouble sending packets (e.g. fatal hardware error).

Example

```
push  ebp                              ;UnBind destroys all registers
push  ebx
movzx ebx, [ebp].CDriverBoardNumber ;EBX has the driver board number
call  LSLUnBindThenDeRegisterMLID   ;UnBind and DeRegister MLID
pop   ebx
pop   ebp
```