Sample Server Driver HSM Template

```
;*****************************************************************************
; NetWare Server Driver HSM Template
;*****************************************************************************


include driver.inc


;*****************************************************************************
; Equates
;*****************************************************************************


TRUE                    equ     1
FALSE                   equ     0

MAX_RETRIES             equ     10
MAX_TIMEOUT_TICKS       equ     36


;(place adapter specific equates here)



;*****************************************************************************
; Structures
;*****************************************************************************


;(place driver required structure definitions here)



;*****************************************************************************
; Multicast Table Structure
;*****************************************************************************

MulticastTableStructure struc

        MulticastAddress        db      6 dup (0)
        EntryUsed               dw      0

MulticastTableStructure ends
```

```
;*****************************************************************************
; Adapter Data Space Structure
;*****************************************************************************

DriverAdapterDataSpace                  struc

   TxInProgress                         dd      FALSE
   TxStartTime                          dd      0
   RetryCounter                         dd      MAX_RETRIES
   InDriverISR                          dd      FALSE
   FirstTimeInit                        dd      TRUE

   LookAheadBuffer                      db      128 dup (0)

  ;(place other adapter specific variables here)

   StatisticsTable                      db      0 dup (?)
      StatisticsVersion                 db      03, 00
      GenericVariableCount              dw      (GenericEnd - GenericBegin) / 4
      CounterMask0                      dd      11111011110011111111011000100011011b

      GenericBegin                      db      0 dup (?)
         TotalTxPacketCount             dd      0               ; 1 - (used by MSM)
         TotalRxPacketCount             dd      0               ; 1 - (used by MSM)
         NoECBAvailableCount            dd      0               ; 1 - (used by MSM)
         PacketTxTooBigCount            dd      0               ; 1 -  not used
         PacketTxTooSmallCount          dd      0               ; 1 -  not used
         PacketRxOverflowCount          dd      0               ; 0 -  used by HSM
         PacketRxTooBigCount            dd      0               ; 1 -  not used
         PacketRxTooSmallCount          dd      0               ; 1 -  not used
         PacketTxMiscErrorCount         dd      0               ; 1 -  not used
         PacketRxMiscErrorCount         dd      0               ; 1 -  not used
         RetryTxCount                   dd      0               ; 0 -  used by HSM
         ChecksumErrorCount             dd      0               ; 0 -  used by HSM
         HardwareRxMismatchCount        dd      0               ; 1 - (used by MSM)
         TotalTxOKByteCountLow          dd      0               ; 1 - (used by MSM)
         TotalTxOKByteCountHigh         dd      0               ; 1 - (used by MSM)
         TotalRxOKByteCountLow          dd      0               ; 1 - (used by MSM)
         TotalRxOKByteCountHigh         dd      0               ; 1 - (used by MSM)
         TotalGroupAddrTxCount          dd      0               ; 1 - (used by MSM)
         TotalGroupAddrRxCount          dd      0               ; 1 - (used by MSM)
         AdapterResetCount              dd      0               ; 0 -  used by HSM
         AdapterOprTimeStamp            dd      0               ; 1 - (used by MSM)
         QDepth                         dd      0               ; 1 - (used by MSM)

         TxOKSingleCollision            dd      0               ; 0 -  used by HSM
         TxOKMultipleCollisions         dd      0               ; 0 -  used by HSM
         TxOKButDeferred                dd      0               ; 1 -  not used
         TxAbortLateCollision           dd      0               ; 0 -  used by HSM
         TxAbortExcessCollisions        dd      0               ; 0 -  used by HSM
         TxAbortCarrierSense            dd      0               ; 0 -  used by HSM
         TxAbortExDeferral              dd      0               ; 1 -  not used
         RxAbortFrameAlignment          dd      0               ; 0 -  used by HSM
      GenericEnd                        db      0 dup (?)

      CustomVariableCount               dw      (CustomEnd - CustomBegin) / 4

      CustomBegin                       db      0 dup (?)
         TransmitTimeoutCount           dd      0
         UnsupportedFrameCount          dd      0
         UnsupportedMulticastCount      dd      0
      CustomEnd                         db      0 dup (?)
      CustomCounterStrings              dd      offset CustomStrings

  ;Align 4 for MOVSD
   AlignDEndVA                  db      (4 - offset (AlignDEndVA and 3)) and 3 dup (?)

DriverAdapterDataSpace                  ends
```

```
        assume  cs: OSCODE, ds: OSDATA, es: OSDATA, ss: OSDATA
OSDATA  segment rw public 'DATA'


;*****************************************************************************
; Driver Parameter Block
;*****************************************************************************

        align   4


DriverParameterBlock            label   dword

DriverParameterSize             dd      DriverParameterBlockSize

DriverStackPointer              dd      0
DriverModuleHandle              dd      0
DriverBoardPointer              dd      0
DriverAdapterPointer            dd      0

DriverConfigTemplatePtr         dd      DriverConfigTemplate

DriverFirmwareSize              dd      0
DriverFirmwareBuffer            dd      0

DriverNumKeywords               dd      0
DriverKeywordText               dd      0
DriverKeywordTextLen            dd      0
DriverProcessKeywordTab         dd      0

DriverAdapterDataSpaceSize      dd      SIZE DriverAdapterDataSpace
DriverAdapterTemplate           dd      DriverAdapterDataSpaceTemplate
DriverStatisticsTable           dd      StatisticsVersion

DriverEndOfChainFlag            dd      0
DriverSendWantsECBs             dd      0
DriverMaxMulticast              dd      -1
DriverNeedsBelow16Meg           dd      0

DriverAESPtr                    dd      0
DriverCallBackPtr               dd      offset DriverCallBack
DriverISRPtr                    dd      offset DriverISR
DriverMulticastChangePtr        dd      offset DriverMulticastChange
DriverPollPtr                   dd      0
DriverResetPtr                  dd      offset DriverReset
DriverSendPtr                   dd      offset DriverSend
DriverShutdownPtr               dd      offset DriverShutdown
DriverTxTimeoutPtr              dd      0
DriverPromiscuousChangePtr      dd      offset DriverPromiscuousChange
DriverStatisticsChangePtr       dd      0
DriverRxLookAheadChangePtr      dd      0
DriverManagementPtr             dd      0
DriverEnableInterruptPtr        dd      offset DriverEnableInterrupt
DriverDisableInterruptPtr       dd      offset DriverDisableInterrupt

DriverParameterBlockSize        equ     $ - DriverParameterBlock
```

```
;*****************************************************************************
; Adapter Data Space Template
;*****************************************************************************

DriverAdapterDataSpaceTemplate  DriverAdapterDataSpace  <>



;*****************************************************************************
; Frame Data Space Template (Configuration Table)
;*****************************************************************************

DriverConfigTemplate    db      0 dup (?)

        db      'HardwareDriverMLID        '    ; [ebx].MLIDCFG_Signature
        db      01                              ; [ebx].MLIDCFG_MajorVersion
        db      12                              ; [ebx].MLIDCFG_MinorVersion
        db      6 dup (0ffh)                    ; [ebx].MLIDNodeAddress
        dw      0010010001001001b               ; [ebx].MLIDModeFlags
        dw      0000                            ; [ebx].MLIDBoardNumber
        dw      0000                            ; [ebx].MLIDBoardInstance
        dd      00000000                        ; [ebx].MLIDMaximumSize
        dd      00000000                        ; [ebx].MLIDMaxRecvSize
        dd      00000000                        ; [ebx].MLIDRecvSize
        dd      00000000                        ; [ebx].MLIDCardName
        dd      DriverNICShortName              ; [ebx].MLIDShortName
        dd      00000000                        ; [ebx].MLIDFrameType
        dw      0000                            ; [ebx].MLIDReserved0
        dw      0000                            ; [ebx].MLIDFrameID
        dw      0001                            ; [ebx].MLIDTransportTime
        dd      000000000                       ; [ebx].MLIDRouteHandler
        dw      10                              ; [ebx].MLIDLineSpeed
        dw      0000                            ; [ebx].MLIDLookAheadSize
        db      8 dup (00h)                     ; [ebx].MLIDReserved1
        db      00                              ; [ebx].MLIDMajorVersion
        db      00                              ; [ebx].MLIDMinorVersion
        dw      0010b                           ; [ebx].MLIDFlags
        dw      0010                            ; [ebx].MLIDSendRetries
        dd      00000000                        ; [ebx].MLIDLink
        dw      0000                            ; [ebx].MLIDSharingFlags
        dw      0000                            ; [ebx].MLIDSlot
        dw      0300h, 32, 0, 0                 ; [ebx].MLIDIOPortsAndLengths
        dd      00000000                        ; [ebx].MLIDMemoryDecode0
        dw      0000                            ; [ebx].MLIDLength0
        dd      00000000                        ; [ebx].MLIDMemoryDecode1
        dw      0000                            ; [ebx].MLIDLength1
        db      03, 0FFh                        ; [ebx].MLIDInterrupt
        db      0FFh, 0FFh                      ; [ebx].MLIDDMAUsage
        dd      00000000                        ; [ebx].MLIDResourceTag
        dd      00000000                        ; [ebx].MLIDConfiguration
        dd      00000000                        ; [ebx].MLIDCommandString
        db      18 dup (0)                      ; [ebx].MLIDLogicalName
        dd      00000000                        ; [ebx].MLIDLinearMemory0
        dd      00000000                        ; [ebx].MLIDLinearMemory1
        dw      0000                            ; [ebx].MLIDChannelNumber
        db      6 dup (0)                       ; [ebx].MLIDIOReserved

Message         DriverNICShortName,     'MYDRIVER'
```

```
;*****************************************************************************
; Parameters required by MSMParseDriverParameters
;*****************************************************************************

IOPort0Data     dd      4
                dd      300h, 320h, 340h, 360h

Interrupt0Data  dd      4
                dd      2, 3, 4, 5

AdapterOptions  AdapterOptionDefinitionStructure <,IOPort0Data,,,,,,,,Interrupt0Data>



;*****************************************************************************
; Example Custom Statistics Strings
;*****************************************************************************

CustomStrings   dw      (EndOfStrings-CustomStrings)

                db      'TransmitTimeoutCount',0
                db      'UnsupportedFramePacketCount',0
                db      'UnsupportedMulticastCount', 0
                db      0,0

EndOfStrings    equ     $



;*****************************************************************************
; Message equates
;*****************************************************************************

CR              equ     13
LF              equ     10



;*****************************************************************************
; Examples of error messages
;*****************************************************************************

TransmitTimeoutMessage  dw       66
                        db      'The cable might be disconnected on the board.'
                        db       CR, LF, 0

HardwareResetFailed     dw       200
                        db      'The adapter failed hardware reset.'
                        db       CR, LF, 0

BufferMemoryFailMessage dw       51
                        db      'Board RAM failed the memory test.'
                        db       CR, LF, 0


OSDATA  ends
```

```
OSCODE  segment er public 'CODE'


;*****************************************************************************
; DriverInit
;*****************************************************************************
;
; Description:  This routine is called by the OS at load time to perform all
;               initialization tasks.
;
; On Entry:     Interrupts are enabled.
;
; On Return:    EAX = 0 if successful
;               Interrupts are preserved
;
;*****************************************************************************


DriverInit      proc


        CPush
        mov     DriverStackPointer, esp                 ; Fill in stack ptr


        ;*************************************************************************
        ; Register HSM and get Frame Data Space
        ;*************************************************************************

        lea     esi, DriverParameterBlock               ; ESI -> Parm Block
        call    EtherTSMRegisterHSM                     ; EBX -> Frame Data Space
        jnz     DriverInitError                         ; Jump if error


        ; EBX -> Frame Data Space (Config Table)


        ;*************************************************************************
        ; Determine Hardware Configuration
        ;*************************************************************************

        mov     eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRESS
        lea     ecx, AdapterOptions
        call    MSMParseDriverParameters
        jnz     DriverInitError


        ;*************************************************************************
        ; Register Hardware Options
        ;*************************************************************************

        call    MSMRegisterHardwareOptions
        cmp     eax, 1                                  ; Error Registering?
        ja      DriverInitError                         ; Jump if so.
        je      DriverInitExit                          ; Skip if new frame.


        ; EBX -> Frame Data Space (Config Table)
        ; EBP -> Adapter Data Space


        ;*************************************************************************
        ; Set Hardware Interrupt
        ;*************************************************************************

        call    MSMSetHardwareInterrupt
        jnz     DriverInitError
```

```
        ;*************************************************************************
        ; Initialize and Test Adapter
        ;*************************************************************************


        ;(initialize adapter specific variables here)


        mov     [ebp].MSMTxFreeCount, 1         ; Allow 1 transmit at a time

        call    DriverReset                     ; Initialize adapter
        jz      short AdapterInitialized        ; jump if success

        push    eax                             ; else save error message
        call    MSMReturnDriverResources        ; return resources
        pop     eax                             ; restore error message
        jmp     DriverInitError


AdapterInitialized:

        mov     [ebp].FirstTimeInit, FALSE      ; Disable DriverReset from
                                                ; testing the hardware again


        ;*************************************************************************
        ; Register the Driver
        ;*************************************************************************

        call    MSMRegisterMLID                 ; Register MLID.
        jnz     short DriverInitError           ; Jump if error.


        ;*************************************************************************
        ; Start Timeout Callbacks
        ;*************************************************************************

        mov     eax, 18                                 ; Schedule 1 sec callbacks
        call    MSMScheduleIntTimeCallBack              ;   to DriverCallBack
        jnz     short DriverInitError                   ; Jump if error


        ;*************************************************************************
        ; Return appropriate status after initialization
        ;*************************************************************************

DriverInitExit:

        xor     eax, eax
        CPop
        ret


DriverInitError:

        mov     esi, eax                        ; ESI -> error message
        call    MSMPrintString                  ; display message

        or      eax, 1                          ; return error status
        CPop
        ret


DriverInit      endp
```

```
;*****************************************************************************
; DriverSend
;*****************************************************************************
;
; Description:  This routine will transfer the packet described in the TCB to
;               the adapter and initiate the send.  TxInProgress, TxStartTime,
;               and RetryCounter must be set for the DriverCallback routine
;               (the transmit timeout watchdog routine).
;
; On Entry:     EBX     => Frame Data Space
;               EBP     => Adapter Data Space
;               ESI     => TCB
;               ECX     Padded Packet Length (Ethernet only)
;
;               Interrupts are disabled and remain disabled.
;
;*****************************************************************************


DriverSend      proc

        ;(adapter specific code to set up board for a send)


        ;*************************************************************************
        ; Copy the Media Header to the NIC
        ;*************************************************************************

        push    esi                             ; save TCB ptr for later
        mov     edi, esi                        ; EDI = -> TCB.

        mov     ecx, [edi].TCBMediaHeaderLen    ; ECX = Media Header Length
        lea     esi, [edi].TCBMediaHeader       ; ESI = -> Media Header


        ;(adapter specific code to copy the Media Header to Transmit Buffer)


        ;*************************************************************************
        ; Copy the Fragments to the NIC
        ;*************************************************************************

        push    ebp                             ; Save Adapter Data Space ptr
        push    ebx                             ; Save Frame Data Space ptr

        mov     edi, [edi].TCBFragStrucPtr      ; EDI = ->Fragment Structure

        mov     ebp, [edi + 0]                  ; EBP = Fragment Count
        or      ebp, ebp                        ; Any fragments?
        jz      EverythingDone                  ; Jump if not

        lea     ebx, [edi + 4]                  ; EBX = ->Fragment Descriptor


CopyFragmentLoop:

        mov     ecx, [ebx].FragmentLength       ; ECX = Fragment Size
        mov     esi, [ebx].FragmentOffset       ; ESI = -> Fragment


        ;(adapter specific code to copy Fragment to Transmit Buffer)


        add     ebx, 8                          ; EBX = -> Next Frag Descriptor
        dec     ebp                             ; More fragments?
        jnz     CopyFragmentLoop                ; Yes; send it.
```

```
EverythingDone:

        pop     ebx                             ; Restore Frame Data Space ptr
        pop     ebp                             ; Restore Adapter Data Space ptr


        ;*************************************************************************
        ; Initiate the transmission
        ;*************************************************************************


        ;(Place adapter specific code here to initiate the transmit)


        ;*************************************************************************
        ; Start transmit timeout monitoring
        ;*************************************************************************

        MSMGetCurrentTime

        mov     [ebp].TxStartTime, eax          ; Starting timer tick for timeout
        mov     [ebp].RetryCounter, MAX_RETRIES ; MAX retries
        mov     [ebp].TxInProgress, TRUE        ; Flag Transmit in Progress


        ;*************************************************************************
        ; Give TCB back to MSM  (Lying Send)
        ;*************************************************************************

        pop     esi                             ; restore TCB pointer

        cmp     [ebp].InDriverISR, 0            ; In DriverISR?
        jnz     EtherTSMSendComplete            ; Jump if so.

        jmp     EtherTSMFastSendComplete        ; Otherwise service events.


DriverSend      endp
```

```
;*****************************************************************************
; DriverCallBack
;*****************************************************************************
;
; Description:  This routine will be executed once every second. It will
;               detect if the hardware does not ack a transmission. If the
;               hardware didn't ack then it will be reset, the transmission
;               of that packet will be aborted and the next packet in the
;               queue will be sent if there is one.
;
; On Entry:     EBX     => Frame Data Space
;               EBP     => Adapter Data Space
;
;               Interrupts are disabled
;
; Remarks:      This routine is called by the MSM at interrupt time.  After
;               this call returns, the MSM will schedule another callback.
;
;*****************************************************************************

DriverCallBack  proc


        ;*************************************************************************
        ; Check if a transmission is in progress
        ;*************************************************************************

        cmp     [ebp].TxInProgress, TRUE        ; Any active transmits?
        jne     DriverCallBackDone              ; Jump if not



        ;*************************************************************************
        ; Is elapsed time greater than maximum transmit time?
        ;*************************************************************************

        MSMGetCurrentTime                       ; EAX = Current Time
        sub     eax, [ebp].TxStartTime          ; EAX = Current Time - TxStartTime.

        cmp     eax, 36                         ; More than 2 seconds passed?
        jl      short DriverCallBackDone        ; Jump if not



        ;*************************************************************************
        ; Determine error if possible and handle, else reset NIC and try again
        ;*************************************************************************


        ;(adapter specific error detection and recovery)


        lea     esi, TransmitTimeoutMessage
        call    MSMAlertWarning
```

```
        ;***************************************************************************
        ; Abort current transmit and reset adapter
        ;***************************************************************************


        ;(adapter specific code to clear board transmit)


        call    DriverReset                     ; Hard reset the card

        mov     [ebp].TxInProgress, FALSE       ; Set Tx no longer active.
        inc     [ebp].MSMTxFreeCount            ; Free up transmit buffer.

        inc     [ebp].TransmitTimeoutCount      ; Update statistics
        inc     [ebp].PacketTxMiscErrorCount    ;


        ;***************************************************************************
        ; Transmit next packet from the send queue
        ;***************************************************************************

        test    [ebp].MSMStatusFlags, TXQUEUED  ; Anything in transmit queue
        jz      short DriverCallBackDone        ; Jump if not

        call    EtherTSMGetNextSend             ; Get next send if any
        jne     short DriverCallBackDone        ; Jump if no send

        push    ebp
        call    DriverSend
        pop     ebp


DriverCallBackDone:

        ret                                     ; Otherwise get out.


DriverCallBack  endp
```

```
;*****************************************************************************
; DriverISR
;*****************************************************************************
;
; Description:  This routine handles packet reception and transmit complete
;               interrupts.
;
; On Entry:     EBP     -> Adapter Data Space
;
;               Interrupts are disabled and remain disabled.
;
;*****************************************************************************


DriverISR       proc


        ;*********************************************************************
        ; Service the interrupt controller
        ;*********************************************************************

        The interrupt controller is normally serviced here. However, if we implement
        the DriverEnableInterrupt and DriverDisableInterrupt routines as recommended,
        the MSM and TSM will call these routines at the appropriate time.


;;;     MSMDisableHardwareInterrupt     ; Do NOT use for Multi-OS support
                                        ;  (see DriverEnableInterrupt)
;;;     MSMDoEndOfInterrupt             ; Do NOT use for Multi-OS support


        mov     [ebp].InDriverISR, TRUE         ; Set for DriverSend.


        ;*********************************************************************
        ; Check Adapter Status (keep returning here until card is fully serviced)
        ;*********************************************************************

CheckStatus:


        ;(adapter specific code to get the adapter status)


        ;if (status == AdapterClear)
                jmp     DriverISRExit

        ;if (status == ReceiveEvent)
                jmp     ReceiveEvent

        ;if (status == TransmitEvent)
                jmp     TransmitEvent
```

```
;*****************************************************************************
; Receive Event Handler
;*****************************************************************************

ReceiveEvent:


        ;(adapter specific code to determine if there are receive errors)

        ;if (ReceiveError)
                jmp     HandleRxErrors


        ;*************************************************************************
        ; Set up a Lookahead Buffer
        ;*************************************************************************


        ;(set up to read in MSMMaxFrameHeaderSize bytes into LookAhead buffer)


        mov     ecx, [ebp].MSMMaxFrameHeaderSize        ; Required Header Size
        lea     edi, [ebp].LookAheadBuffer              ; -> LookAhead buffer


        ;(adapter specific code to read into LookAhead buffer)



        ;*************************************************************************
        ; Get an RCB
        ;*************************************************************************

        lea     esi, [ebp].LookAheadBuffer
        ;(mov   ecx, PacketSize)
        ;(mov   eax, StatusOfReceivedPacket)
        call    EtherTSMGetRCB
        jnz     NoRCBAvailable


        ; ESI = -> RCB
        ; EDI = -> Fragment Structure
        ; EBX = Bytes in packet to skip over.
        ; ECX = Bytes remaining to be read from NIC.



        ;*************************************************************************
        ; Copy packet data into RCB fragments
        ;*************************************************************************

        push    ebp                             ; save Adapter Data Space ptr

        ;(adapter specific code to set up for reading packet into RCB Fragments)

        mov     ebp, [edi + 0]                  ; EBP = Fragment Count
        lea     ebx, [edi + 4]                  ; EDI = -> Fragment Descriptor
```

```
ReadFragmentLoop:

        mov     edi, [ebx].FragmentOffset       ; EDI -> Fragment offset
        mov     ecx, [ebx].FragmentLength       ; ECX = Fragment size

        ;(adapter specific code to read data into fragment)

        add     ebx, 8                          ; EBX = -> Next Frag Descriptor
        dec     ebp                             ; Decrement fragment count
        jnz     ReadFragmentLoop                ; Jump if more fragments


        pop     ebp                             ; Restore Adapter Data Space ptr



        ;*****************************************************************************
        ; Give RCB back to MSM
        ;*****************************************************************************

        call    EtherTSMRcvComplete             ; Give RCB back


FinishUp:

        ;(adapter specific code to set up board for next receive)

        jmp     CheckStatus



        ;*****************************************************************************
        ; Unable to receive packet. Increment proper statistic counter.
        ;*****************************************************************************

NoRCBAvailable:

        inc     [ebp].UnsupportedFrameCount     ; Update statistics
        jmp     FinishUp                        ; Skip receiving this packet.



        ;*****************************************************************************
        ; Handle Receive Errors
        ;*****************************************************************************

HandleRxErrors:


        ;(adapter specific code to determine and handle receive errors)
        ;(make sure you update the appropriate statistics counters)

        jmp     CheckStatus
```

```
;*****************************************************************************
; Transmit Event Handler
;*****************************************************************************

TransmitEvent:

        ;if (Transmit Error)
                jmp  TransmitError


TransmitComplete:

        ;*************************************************************************
        ; Transmit next packet from the MSM's send queue
        ;*************************************************************************

TransmitNextPacket:

        mov     [ebp].TxInProgress, FALSE       ; Transmit not in progress
        inc     [ebp].MSMTxFreeCount            ; Free up transmit buffer

        test    [ebp].MSMStatusFlags, TXQUEUED  ; Anything in send queue
        jz      CheckStatus                     ; If not, check adapter status

        call    EtherTSMGetNextSend             ; Get next TCB queued
        jnz     CheckStatus                     ; If none, check adapter status

        call    DriverSend                      ; Send TCB
        jmp     CheckStatus                     ; check adapter status



        ;*************************************************************************
        ; Attempt to retry this transmission
        ;*************************************************************************

TransmitError:


        ;(adapter specific code to determine and handle transmit errors)


        dec     [ebp].RetryCounter              ; Any more retries?
        jz      TransmitNextPacket              ; Give up if not
        inc     [ebp].RetryTxCount              ; Update statistics


        ;(adapter specific code to re-start the transmit)


        MSMGetCurrentTime                       ; EAX = current time
        mov     [ebp].TxStartTime, eax          ; Record transmit start time
        mov     [ebp].TxInProgress, TRUE        ; Flag transmit in progress
        jmp     CheckStatus                     ; Check status again



DriverISRExit:

;;;     MSMEnableHardwareInterrupt      ; Do NOT use for Multi-OS support
                                        ; (see DriverEnableInterrupt)

        mov     [ebp].InDriverISR, FALSE        ; Not in ISR
        MSMServiceEventsAndRet                  ; Let LSL service events



DriverISR       endp
```

```
;*****************************************************************************
; DriverEnableInterrupt
;*****************************************************************************
;
; Description:  This routine will enable interrupts at the adapter hardware
;
; On Entry:     EBP     -> Adapter Data Space
;
; On Return:    EBP     must be preserved
;
;*****************************************************************************

DriverEnableInterrupt   proc

;;      (adapter specific code to enable interrupts at the adapter hardware)

        ret

DriverEnableInterrupt   endp




;*****************************************************************************
; DriverDisableInterrupt
;*****************************************************************************
;
; Description:  This routine will disable interrupts at the adapter hardware
;
; On Entry:     EBP     -> Adapter Data Space
;
; On Return:    EBP     must be preserved
;
;       Adapters/drivers that do not share interrupts:
;               EAX = zero
;
;       Adapters/drivers that share interrupts:
;               EAX = zero if the interrupt was from our board
;               EAX = non-zero if the interrupt was not from our board
;       (in this last case, the TSM will immediately call DriverEnableInterrupt)
;
;*****************************************************************************

DriverDisableInterrupt  proc

;;      (adapter specific code to disable interrupts at the adapter hardware)

;;      (mov    eax, Appropriate Status)

        ret

DriverDisableInterrupt  endp
```

```
;*****************************************************************************
; DriverMulticastChange
;*****************************************************************************
;
; Description:  This routine will modify the NIC's multicast registers to enable
;               it to receive the multicast addresses listed in the multicast
;               table.  Each entry in the multicast table is as follows:
;
;               bytes 0-5 = Multicast Address.
;               bytes 6-7 = Entry used (Non zero if used).
;
; On Entry:     EBP     -> Adapter Data Space
;               EBX     -> Frame Data Space
;               ESI     -> Multicast Table (default for Ethernet & FDDI)
;               ECX     # of Entries in Table (default for Ethernet & FDDI)
;               EDX     32-bit functional addr (default for Token-Ring & PCN2)
;
; On Return:    EBX     Preserved
;               EBP     Preserved
;
;*****************************************************************************


DriverMulticastChange   proc


        ;*************************************************************************
        ; Reset Multicast Address Registers
        ;*************************************************************************

        ;(adapter specific code to clear current Multicast settings)


        ;*************************************************************************
        ; Loop thru each MSM table entry and set current multicast addresses
        ;*************************************************************************

        or      ecx, ecx
        jz      short MulticastChangeExit       ; Leave if table is empty.


ChangeMulticastLoop:

        push    ecx                             ; Save loop counter.

        ; [esi+0].MulticastAddress
        ; [esi+6].EntryUsed


        ;(adapter specific code to update board for this Multicast address entry)


        pop     ecx                             ; Restore loop counter.


NextMulticastEntry:

        add     esi, SIZE MulticastTableStructure       ; Get next entry.
        dec     ecx
        jnz     short ChangeMulticastLoop


MulticastChangeExit:


        ;(adapter specific code to enable NIC receives)

        ret


DriverMulticastChange   endp
```

```
;*****************************************************************************
; DriverPromiscuousChange
;*****************************************************************************
;
; Description:  This routine will enable/disable the Promiscuous Mode.
;
; On Entry:     EBP     -> Adapter Data Space
;               EBX     -> Frame Data Space
;               ECX     0 to disable the Promiscuous mode
;
; On Return:    EBX     Preserved
;               EBP     Preserved
;
;*****************************************************************************


DriverPromiscuousChange proc


        or      ecx, ecx                        ; Disable Promiscuous?
        jz      short DriverPromiscuousDisable


DriverPromiscuousEnable:

        ;(adapter specific code to enable promiscuous mode reception)

        ret


DriverPromiscuousDisable:

        ;(adapter specific code to disable promiscuous mode reception)

        ret


DriverPromiscuousChange endp
```

```
;*****************************************************************************
; DriverReset
;*****************************************************************************
;
; Description:  This routine will reset and initialize the NIC.
;
; On Entry:     EBX     -> Frame Data Space
;               EBP     -> Adapter Data Space
;
;               Interrupts are disabled
;
; On Return:    EAX     0 if successful (otherwise points to error message)
;               EBX     Preserved
;               EBP     Preserved
;
;               Interrupts preserved.
;
;*****************************************************************************


DriverReset     proc    near


        ;(hard reset the adapter)


        ;*************************************************************************
        ; If this is the first time through, then do some hardware tests and any
        ; initialization that need only be done once.
        ;*************************************************************************

        cmp     [ebp].FirstTimeInit, TRUE       ; First time thru?
        jne     short SkipFirstTimeInit         ; Skip if not.

        ;(call adapter specific proc to test hardware and do 1st time inits)

        or      eax, eax                        ; Error message?
        jnz     short DriverResetError          ; Jump if so.


SkipFirstTimeInit:

        call    EtherTSMUpdateMulticast         ; Init Multicast registers.

        ;(adapter specific code to start NIC and verify operational)


DriverResetSuccess:

        xor     eax, eax
        ret


DriverResetError:

        ;(adapter specific code to make sure adapter is safely shutdown)

        or      eax, eax                ;return error message
        ret


DriverReset     endp
```

```
;*****************************************************************************
; DriverShutdown
;*****************************************************************************
;
; Description:  This routine will shutdown the adapter.
;
; On Entry:     EBX     -> Frame Data Space
;               EBP     -> Adapter Data Space
;               ECX     0 if Permanent Shutdown
;
;               Interrupts disabled.
;
; On Return:    EAX     0 if successful
;
;               Interrupts preserved.
;
;*****************************************************************************


DriverShutdown  proc


        or      ecx, ecx
        jnz     PartialShutdown


        ;(return any dynamically allocated memory using MSMFree or MSMFreePages)


PartialShutdown:

        ;(return any preallocted RCBs using MSMReturnRCB)
        ;(return any queued TCBs using <TSM>SendComplete/MSMServiceEvents)

        ;(shutdown the adapter)


DriverShutdownSuccess:

        xor     eax, eax                        ; Good Return code.
        ret


DriverShutdownFailed:

        mov     eax, 0FFFFFF84h
        or      eax, eax
        ret


DriverShutdown  endp
```

```
;*****************************************************************************
; DriverRemove
;*****************************************************************************
;
; Description:  This routine calls the MSM to return our resources.
;
; On Entry:     Interrupts can be in any state
; On Return:    Interrupts are preserved
;
; Remarks:      This routine is called by the OS at unload.
;               It is called at process time.
;
;*****************************************************************************


DriverRemove    proc


        CPush
        mov     eax, DriverModuleHandle
        call    MSMDriverRemove
        CPop
        ret


DriverRemove    endp







OSCODE  ends

        end
```