

Appendix E

MSL Driver Template

DriverInitialize	E-7
HardwareInit	E-11
DriverControl	E-12
GetMSLConfiguration - Driver control function 0	E-13
GetMSLStatistics - Driver control function 1	E-13
DriverSend	E-14
DriverBuildSend	E-15
DriverEmergencySend	E-16
DriverISR	E-17
General Packet Reception Handler	E-18
Message Packet Received	E-18
Acknowledgement Received	E-22
Hold Notification Received	E-23
Emergency Notification Received	E-23
Transmit Complete / Transmit Error	E-24
TransmitMessagePacket	E-25
TransmitAcknowledgement	E-26
TransmitHoldNotification	E-27
DriverTimeOut	E-28
DriverHoldOff	E-30
DriverIntHoldOff	E-31
HoldOffDeliverMessageToOS	E-32
DriverRemove	E-35

This appendix contains a listing of a sample MSL driver template for an adapter with a single communications channel. The template code presented is intended to illustrate a general flow of events and does not necessarily describe optimized code, nor does it apply in every case.


```

;*****
;*
;*                               MSL Driver Template                               *
;*                               ( with single communications channel )                *
;******
;*****
;* Include Files *
;*****

include MSL.inc                               ;MSL include file

;*****
;* MSL Template Equates *
;*****

TIMEOUT_CALLBACK_INTERVAL      equ      1

ADAPTER_TIMEOUT_COUNT         equ      5          ;(adapter specific)
MESSAGE_TIMEOUT_COUNT         equ      5          ;(adapter specific)

MAX_EMERGENCY_WAIT            equ      54         ;(adapter specific)
MAX_PACKET_SIZE               equ      4096       ;(adapter specific)

MESSAGE_TYPE                   equ      -1        ;packet types
ACK_TYPE                       equ      0
HOLDOFF_TYPE                   equ      1
EMERGENCY_TYPE                 equ      2

;*****
;* MSL Template Structures *
;*****

MESSAGE_HEADER_SIZE           equ      SIZE      MessageHeaderStructure

MessageHeaderStructure        struc                ;example only
    EaxParameter              dd      ?
    EbxParameter              dd      ?
    EcxParameter              dd      ?
    EdxParameter              dd      ?
    EsiParameter              dd      ?
    EdiParameter              dd      ?
MessageHeaderStructure        ends

PACKET_HEADER_SIZE            equ      SIZE      PacketHeaderStructure

PacketHeaderStructure         struc                ;example only
    MediaHeader                dd      7 dup (?)
    MSLType                    db      ?
    MSLMessageCount            db      ?
    MSLSequence                dd      ?
PacketHeaderStructure         ends

```

```

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

```

```

          extrn   MaximumCommDriverDataLength: dword
          extrn   PacketSizeNowAvailable: dword
          extrn   PacketSizeDriverCanNowHandle: dword
          extrn   ServerCommACKTimeOut: dword

          extrn   GetNextPacketPointer: dword
          extrn   ReceiveServerCommPointer: dword
          extrn   SendServerCommCompletedPointer: dword

```

```
Align 16
```

```

;*****
;* Resource Tag Information *
;*****

```

```

RTagMessage_AESProcess      db      'AES Callback'           , 0
RTagMessage_Timer           db      'Timer Callback'         , 0
RTagMessage_IORegistration  db      'Hardware Config'        , 0
RTagMessage_Interrupt       db      'Hardware ISR'           , 0
RTagMessage_MSL             db      'MSL driver name'         , 0

```

```

InterruptResourceTag        dd      0
MSLDriverResourceTag        dd      0

```

```

;*****
;* Initialization Error Messages *
;*****

```

```

ErrorGettingRTag_Interrupt  db      'Unable to allocate Interrupt resource tag'
                             db      LF, CR, BELL, 0

ErrorGettingRTag_Timer      db      'Unable to allocate Timer Event resource tag'
                             db      CR, LF, BELL, 0

ErrorGettingRTag_AESProcess db      'Unable to allocate AES Process resource tag'
                             db      CR, LF, BELL, 0

ErrorGettingRTag_IORegistration db  'Unable to allocate Hardware Options resource tag'
                             db      CR, LF, BELL, 0

ErrorGettingRTag_MSL        db      'Unable to allocate MSL resource tag'
                             db      CR, LF, BELL, 0

ErrorParsingIOMessage       db      'Error parsing IO Parameters.'
                             db      CR, LF, BELL, 0

ConflictingHardwareMessage  db      'Conflicting Hardware Options.'
                             db      CR, LF, BELL, 0

ErrorGettingInterruptMessage db  'Error setting interrupt.'
                             db      CR, LF, BELL, 0

ErrorRegisteringMSLMessage  db      'Unable to register MSL with OS'
                             db      CR, LF, BELL, 0

HardwareInitErrorMessage    db      'Error initializing adapter hardware'
                             db      CR, LF, BELL, 0

```

```

;*****
;* Configuration Information *
;*****

Align 16

IOPort0_Options          dd      4
                        dd      300h, 320h, 340h, 360h

Interrupt0_Options       dd      4
                        dd      2, 3, 4, 5

AdapterOptions  AdapterOptionStructure  <,IOPort0_Options,,,,,,,,,Interrupt0_Options>

DriverConfiguration      IOConfigurationStructure  <>
DriverConfigurationSize  equ      SIZE      IOConfigurationStructure

;*****
;* Statistics Information *
;*****

DriverStatistics         db      0 dup (?)

    StatisticsMajorVersion  db      01
    StatisticsMinorVersion  db      00
    NumGenericCounters      dw      (GenericEnd - GenericBegin) / 4
    NotSupportedMask        dd      00000000000000000001111111111111b
    GenericBegin            db      0 dup (?)
        TransmitPacketCount  dd      0
        ReceivePacketCount   dd      0
        TransmitBurstPacketCount  dd      0
        MSLRejectPacketCount  dd      0
        TransmitMsgCount     dd      0
        ReceiveAckCount      dd      0
        ReceiveMsgCount      dd      0
        TransmitAckCount     dd      0
        TransmitHoldCount    dd      0
        ReceiveHoldCount     dd      0
        OSHoldMsgCount       dd      0
        OSCallBackCount      dd      0
        OSRejectMsgCount     dd      0
        ServerCommErrorCount  dd      0
        ReceiveErrorCount    dd      0
        TransmitErrorCount   dd      0
        ReceiveEmergencyCount  dd      0
        RetryTxCount         dd      0
    GenericEnd              db      0 dup (?)

    NumCustomCounters       dw      (CustomEnd - CustomBegin) / 4
    CustomBegin             db      0 dup (?)
        MessageTimedOutCount  dd      0
        AdapterTimedOutCount  dd      0
    CustomEnd               db      0 dup (?)
    CustomStrings           db      0 dup (?)
    CustomStringsSize       dw      (CustomStringsEnd-CustomStrings)
                        db      'MessageTimedOutCount', 0
                        db      'AdapterTimedOutCount', 0
                        db      0,0
    CustomStringsEnd        db      0 dup (?)

DriverStatisticsEnd        db      0 dup (?)

DriverStatisticsSize       equ      DriverStatisticsEnd - DriverStatistics

```

```

;*****
;* Timeout and Timer Data Variables *
;*****

TimeOutEvent      AESEventStructure      <,TIMEOUT_CALLBACK_INTERVAL,,DriverTimeOut>
HoldOffEvent      AESEventStructure      <,0,,DriverHoldOff>
IntHoldOffEvent   TimerDataStructure     <,DriverIntHoldOff,,1>

;*****
;* General Data Variables *
;*****

PacketHeader      PacketHeaderStructure  <>
MessageHeader     MessageHeaderStructure  <>

Align 16

NewEsiParameter   dd      0
NewEcxParameter   dd      0

ExtraEOIFlag      dd      0
FirstTimeInit     dd      0                ;indicate reset or not

RxPacketMessageCount  dd      0                ;num of messages in Rx packet
TxPacketMessageCount  dd      0                ;num of messages in Tx packet

;*** Transmit pending flags (pending due to another transmit in progress) ***
MessageTransmitPending  dd      0                ;Message Tx pending flag
AckTransmitPending      dd      0                ;Ack Tx pending flag
HoldTransmitPending     dd      0                ;Hold Tx pending flag

;*** Error notification pending flags (pending due to being in a holdoff state) ***
EmergencyReceivePending  dd      0                ;Emergency Rx pending flag
TimeOutErrorPending      dd      0                ;TimeOut Error pending flag
HardwareErrorPending     dd      0                ;Hardware Error pending flag

TransmitInProgress      dd      0                ;flag packet sending
MessageInProgress       dd      0                ;flag of message send

```

```

;*****
;* Receive buffers and redeliver variables *
;*****

CurrentReceiveBufferAddress      dd      ReceiveBuffer1
CurrentReceiveBufferFillPtr      dd      0                ;current end of copy data
                                   ;in receive buffer

CurrentReceiveHoldBufferAdd      dd      0
CurrentHoldBufferMsgPointer      dd      0

CurrentBufferIndex               dd      0
CurrentHoldBufferIndex           dd      0

CurrentReceiveBufferTable        dd      ReceiveBuffer1
                                   dd      ReceiveBuffer2
                                   dd      ReceiveBuffer3

ReceiveBuffer1                   db      (4096 + PACKET_HEADER_SIZE) dup (0)
ReceiveBuffer2                   db      (4096 + PACKET_HEADER_SIZE) dup (0)
ReceiveBuffer3                   db      (4096 + PACKET_HEADER_SIZE) dup (0)

HoldNewEsiParameter              dd      0
HoldNewEcxParameter              dd      0

HoldStateFlag                    dd      0                ;in holdoff indicator
LastHoldTransmitTime             dd      0

BackOffAmount                    dd      0                ;for next send if Hold
HoldOffWaitLoopCount             dd      0                ;for current send

OSDATA ends

```

```

OSCODE segment er public 'CODE'

    public DriverInitialize           ;MSL driver routine
    public DriverSend                 ;MSL driver routine
    public DriverBuildSend           ;MSL driver routine
    public DriverISR                  ;MSL driver routine
    public DriverRemove               ;MSL driver routine

    extrn AllocateResourceTag: near   ;OS routine
    extrn CancelInterruptTimeCallBack: near ;OS routine
    extrn CancelNoSleepAESProcessEvent: near ;OS routine
    extrn CancelSleepAESProcessEvent: near ;OS routine
    extrn ClearHardwareInterrupt: near ;OS routine
    extrn CRescheduleLast: near       ;OS routine
    extrn DeRegisterHardwareOptions: near ;OS routine
    extrn DeRegisterServerCommDriver: near ;OS routine
    extrn GetCurrentTime: near        ;OS routine
    extrn OutputToScreen: near        ;OS routine
    extrn ParseDriverParameters: near ;OS routine
    extrn RegisterHardwareOptions: near ;OS routine
    extrn RegisterServerCommDriver: near ;OS routine
    extrn ScheduleInterruptTimeCallBack: near ;OS routine
    extrn ScheduleNoSleepAESProcessEvent: near ;OS routine
    extrn ScheduleSleepAESProcessEvent: near ;OS routine
    extrn ServerCommDriverError: near ;OS routine
    extrn SetHardwareInterrupt: near ;OS routine

Align 16

;*****
;* Control Procedure Vector Information *
;*****

ControlProcedures      dd      GetMSLConfiguration
                       dd      GetMSLStatistics
ControlProceduresEnd   equ     $
MaxControlNumber       equ     ((ControlProceduresEnd-ControlProcedures)/4)-1

```



```

;*****
;* DriverInitialize
;*****
;*
;* Stack Parameters:
;*
;* Parm0 = ModuleHandle          Parm5 = LoadableModuleFileHandle
;* Parm1 = ScreenHandle          Parm6 = ReadRoutine
;* Parm2 = CommandLine           Parm7 = CustomDataOffset
;* Parm3 = (reserved)           Parm8 = CustomDataSize
;* Parm4 = (reserved)
;*
;*****

Align 16
DriverInitialize      proc

    CPush
    mov     ebp, esp
    pushfd
    cli

;*****
;* Allocate all resource tags used by MSL
;*****

    push   MSLSignature
    push   OFFSET RTagMessage_MSL
    push   [ebp + Parm0]
    call   AllocateResourceTag
    add    esp, 3 * 4
    or     eax, eax
    mov    MSLDriverResourceTag, eax
    mov    ebx, OFFSET ErrorGettingRTag_MSL
    jz     DisplayMessageExit

    push   IORegistrationSignature
    push   OFFSET RTagMessage_IORegistration
    push   [ebp + Parm0]
    call   AllocateResourceTag
    add    esp, 3 * 4
    or     eax, eax
    mov    DriverConfiguration.CIOResourceTag, eax
    mov    ebx, OFFSET ErrorGettingRTag_IORegistration
    jz     DisplayMessageExit

    push   InterruptSignature
    push   OFFSET RTagMessage_Interrupt
    push   [ebp + Parm0]
    call   AllocateResourceTag
    add    esp, 3 * 4
    or     eax, eax
    mov    InterruptResourceTag, eax
    mov    ebx, OFFSET ErrorGettingRTag_Interrupt
    jz     DisplayMessageExit

    push   TimerSignature
    push   OFFSET RTagMessage_Timer
    push   [ebp + Parm0]
    call   AllocateResourceTag
    add    esp, 3 * 4
    or     eax, eax
    mov    IntHoldOffEvent.TResourceTag, eax
    mov    ebx, OFFSET ErrorGettingRTag_Timer
    jz     DisplayMessageExit

```

```

push    AESProcessSignature
push    OFFSET RTagMessage_AESProcess
push    [ebp + Parm0]
call    AllocateResourceTag
add     esp, 3 * 4
or      eax, eax
mov     TimeOutEvent.AESRTag, eax
mov     HoldOffEvent.AESRTag, eax
mov     ebx, OFFSET ErrorGettingRTag_AESProcess
jz     DisplayMessageExit

;*****
;* Parse which port and interrupt to use *
;*****

push    [ebp + Parm1]
push    [ebp + Parm2]
push    NeedsIOPort0Bit OR NeedsInterrupt0Bit
push    0
push    0
push    OFFSET AdapterOptions
push    0
push    OFFSET DriverConfiguration
call    ParseDriverParameters
add     esp, 8 * 4
or      eax, eax
mov     ebx, OFFSET ErrorParsingIOMessage
jnz    DisplayMessageExit

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

push    0
push    OFFSET DriverConfiguration
call    RegisterHardwareOptions
add     esp, 2 * 4
or      eax, eax
mov     ebx, OFFSET ConflictingHardwareMessage
jnz    DisplayMessageExit

;*****
;* Set Interrupt Vector *
;*****

push    OFFSET ExtraEOIFlag
push    CHAIN_SET_REAL_MODE
push    0
push    InterruptResourceTag
push    OFFSET DriverISR
movzx   eax, BYTE PTR DriverConfiguration.CInterrupt0
push    eax
call    SetHardwareInterrupt
add     esp, 6 * 4
or      eax, eax
mov     ebx, OFFSET ErrorGettingInterruptMessage
jnz    DeRegisterHardware

;*****
;* Initialize and Test the MSL Adapter *
;*****

call    HardwareInit                ;returns ptr to error message
jnz    DriverInitHardwareError
mov     FirstTimeInit, 0            ;disable testing the adapter
                                           ;hardware again

```

```

;*****
;* Register the MSL driver with the OS
;*****

push    OFFSET DriverControl
push    OFFSET DriverEmergencySend
push    OFFSET DriverBuildSend
push    OFFSET DriverSend
push    OFFSET DriverConfiguration
push    MSLDriverResourceTag
call    RegisterServerCommDriver
add     esp, 6 * 4
or      eax, eax
mov     ebx, OFFSET ErrorRegisteringMSLMessage
jnz     ErrorRegisteringDriver

mov     MaximumCommDriverDataLength, MAX_PACKET_SIZE
mov     PacketSizeDriverCanNowHandle, MAX_PACKET_SIZE

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

push    OFFSET TimeOutEvent
call    ScheduleNoSleepAESProcessEvent
add     esp, 1 * 4

;*****
;* DriverInitialize Successful Exit
;*****

popfd
xor     eax, eax
CPop
ret
; Return Success!

```

```

;*****
;* DriverInitialize Error Paths
;*****

ErrorRegisteringDriver:
DriverInitHardwareError:

;*****
;* Unhook from Interrupt vector
;*****

push    OFFSET DriverISR
movzx   eax, BYTE PTR DriverConfiguration.CInterrupt0
push    eax
call    ClearHardwareInterrupt
add     esp, 2 * 4

;*****
;* Deregister hardware options from OS
;*****

DeRegisterHardware:

push    OFFSET DriverConfiguration
call    DeRegisterHardwareOptions
add     esp, 1 * 4

;*****
;* Display Error Message in EBX
;*****

DisplayMessageExit:

push    ebx                ; Pointer to error string
push    [ebp + Parm1]      ; Screen Handle
call    OutputToScreen     ; Display Error Message
add     esp, 2 * 4

popfd
or      eax, -1            ; Return Failure
CPop
ret

DriverInitialize          endp

```

```
;*****  
;* HardwareInit *  
;*****  
  
HardwareInit    proc  
  
                ; (Adapter-specific code to bring up adapter to operational mode)  
  
HardwareInitSuccess:  
                xor     eax, eax  
                ret  
  
HardwareInitError:  
                mov     ebx, OFFSET HardwareInitErrorMessage  
                or     eax, -1  
                ret  
  
HardwareInit    endp
```

```

;*****
;*  DriverControl
;*****
;*
;*  Function 0 = GetMSLConfiguration
;*  Function 1 = GetMSLStatistics
;*
;*  Stack Parameters:
;*
;*      Parm0 = Function Number
;*      Parm1 = Pointer to the buffer to copy Configuration or Statistics.
;*              If pointer=0, return size of Configuration or Statistics.
;*
;*****

Align 16
DriverControl  proc

    CPush                ;save C registers
    mov     ebp, esp     ;get stack base
    pushfd                ;save flag state
    cli                 ;clear interrupts

    mov     ebx, [ebp + Parm0] ;get requested function #
    cmp     ebx, MaxControlNumber ;check if request is valid
    ja     InvalidControlProcedure ;jump if not

    call    ControlProcedures [ebx * 4] ;table thru routine

DriverControlExit:

    popfd                ;restore flags
    CPop                 ;restore registers
    ret

InvalidControlProcedure:

    mov     eax, BAD_COMMAND ;flag invalid status
    jmp     DriverControlExit

DriverControl  endp

```

```

;*****
;* GetMSLConfiguration - Driver control function 0 *
;*****

Align 16
GetMSLConfiguration      proc

        mov     edi, [ebp + Parm1]           ;get buffer pointer
        or      edi, edi                     ;get size only? (edi=0)
        jz      SHORT GetMSLConfigurationSize ;jump if so

        mov     ecx, DriverConfigurationSize ;copy the configuration
        mov     esi, OFFSET DriverConfiguration
rep     movsb
        xor     eax, eax
        ret

GetMSLConfigurationSize:

        mov     eax, DriverConfigurationSize ;get configuration size
        ret

GetMSLConfiguration      endp

;*****
;* GetMSLStatistics - Driver control function 1 *
;*****

Align 16
GetMSLStatistics         proc

        mov     edi, [ebp + Parm1]           ;get buffer pointer
        or      edi, edi                     ;get size only? (edi=0)
        jz      SHORT GetMSLStatisticsSize  ;jump if so

        mov     ecx, DriverStatisticsSize   ;copy the statistics
        mov     esi, OFFSET DriverStatistics
rep     movsb
        xor     eax, eax
        ret

GetMSLStatisticsSize:

        mov     eax, DriverStatisticsSize   ;get statistics size
        ret

GetMSLStatistics         endp

```

```

;*****
;* DriverSend
;*****
;*
;*      On Entry:                                On Exit:
;*
;*      EAX = OS parameter                        EAX = Not saved
;*      EBX = OS parameter                        EBX = Not saved
;*      ECX = OS parameter/Length of Message Data ECX = Not saved
;*      EDX = OS parameter                        EDX = Not saved
;*      EBP = Not Defined                         EBP = Not saved
;*      ESI = OS parameter/Pointer to Message Data ESI = Not saved
;*      EDI = OS parameter                        EDI = Not saved
;*
;*      Interrupts Disabled
;*
;*****

Align 16
DriverSend      proc

        mov     PacketSizeDriverCanNowHandle, -1      ;inform OS we're busy
        mov     TxPacketMessageCount, 1              ;sending one message
        inc     TransmitMsgCount                     ;update statistics counter

;*****
;* Build Message Packet in Transmit Buffer
;*****

; Note: this code assumes the hardware can accept loading of message data
; even if the channel is busy with another transmit

        (Setup packet header and message header in transmit buffer here)

        or      ecx, ecx                             ;any data with message?
        jz      DriverSendReady                     ;skip data copy if not

        (Copy message data to transmit buffer here:  ecx=size  esi=addr)

DriverSendReady:

        call    TransmitMessagePacket
        inc     TransmitPacketCount
        xor     eax, eax
        ret

DriverSend      endp

```



```

;*****
;* DriverBuildSend
;*****
;*
;*      On Entry:                                On Exit:
;*
;*      EAX = OS parameter                        EAX = Not saved
;*      EBX = OS parameter                        EBX = Not saved
;*      ECX = OS parameter/Length of Message Data ECX = Not saved
;*      EDX = OS parameter                        EDX = Not saved
;*      EBP = Not Defined                        EBP = Not saved
;*      ESI = OS parameter/Pointer to Message Data ESI = Not saved
;*      EDI = OS parameter                        EDI = Not saved
;*
;*      Interrupts Disabled
;*
;*****

MAX_MESSAGE_COUNT      equ      128                ;maximum # of messages in
                                                ;a multi-message packet
                                                ;(optimize for your design)

Align 16
DriverBuildSend proc

    (Build Message Header)

    or     ecx, ecx                                ;any data with message
    jz     DriverBuildSendDone                    ;skip data copy if not

    (Copy Message Data to adapter:  ecx=size  esi=addr)

DriverBuildSendDone:

    inc    TransmitMsgCount                       ;update statistics counter
    inc    TxPacketMessageCount                   ;update message count

    cmp    TxPacketMessageCount, MAX_MESSAGE_COUNT ;max # of messages yet?
    je     HitMaxMessageCount                     ;jump if so

    sub    PacketSizeDriverCanNowHandle, MESSAGE_HEADER_SIZE
    sub    PacketSizeDriverCanNowHandle, ecx

    xor    eax, eax                               ;indicate success
    ret                                         ;return to OS

HitMaxMessageCount:

    mov    PacketSizeDriverCanNowHandle, -1       ;indicate no more msgs
                                                ; for this packet

    xor    eax, eax
    ret                                         ;return to OS

DriverBuildSend endp

```

```
*****
;* DriverEmergencySend
*****

Align 16
DriverEmergencySend    proc

    cmp     TransmitInProgress, FALSE        ;if not transmitting now
    je      EmergencySendReady              ;fire Emergency packet

    mov     ecx, MAX_EMERGENCY_WAIT         ;set adapter specific wait

EmergencySendWaitLoop:

    in      al, 61h                          ;else waste time
    in      al, 61h                          ;(same for all speed machines)

    (Read Adapter Status)
    (If transmit channel is now available...jmp EmergencySendReady)

    loop   EmergencySendWaitLoop

ForceEmergencySend:

    (If possible, cancel the adapter's current transmission and force send)

EmergencySendReady:

    (Transmit the Emergency Notification)

    ret

DriverEmergencySend    endp
```

```

;*****
;* DriverISR
;*****

Align 16
DriverISR      proc

    (mask off adapter's interrupt(s))

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

    mov     al, EOI
    cmp     ExtraEOIFlag, 0
    jz      SHORT SkipExtraEOI
    out     ATInterruptCtrlRegister, al

SkipExtraEOI:

    out     InterruptCtrlRegister, al

;*****
;* Check Adapter Status
;*****

CheckAdapterStatus:

    (get the adapter status)

; Note: The order of parsing the cause of interrupt may be adapter
; specific so the following order may change for your adapter.

    if (status==AdapterClear)      jmp     ISRExit
    if (status==ReceiveEvent)     jmp     ISRReceiveEvent
    if (status==TransmitEvent)    jmp     ISRTransmitEvent

```

```

;*****
;* General Packet Reception Handler *
;*****

ISRReceiveEvent:

    (validate error free reception of packet)

    mov     eax, CurrentReceiveBufferAddress    ;setup incase of hold
    mov     CurrentReceiveBufferFillPtr, eax

    (read Packet Header and MSL Header into one of three receive buffers)

    cmp     PacketHeader.MSLType, MESSAGE_TYPE    ;Message packet?
    je     ISRMessagePacketReceived

    cmp     PacketHeader.MSLType, EMERGENCY_TYPE    ;Emergency notification?
    je     ISREmergencyReceived

    cmp     PacketHeader.MSLType, HOLDOFF_TYPE    ;HoldOff notification?
    je     ISRHoldReceived

    jmp     ISRAckReceived                        ;must be Acknowledgement

;*****
;* Message Packet Received *
;*****

;*****
;* If previous message(s) are being held off by the OS, copy entire message *
;* packet into next free receive buffer and deliver its message(s) after the *
;* current held off message(s). Note: In most adapters the MSL driver would *
;* not acknowledge the messages, thus stopping the flow of data message(s). *
;*****

ISRMessagePacketReceived:

    inc     ReceivePacketCount

    cmp     HoldStateFlag, 0                    ;if last message pack held...
    jne     ISRHoldOffMessage                  ;...hold this one

    call    TransmitAcknowledgement

ISRReadMessageCount:

;*****
;* Get and save the number of messages in the packet *
;*****

    movzx   eax, PacketHeader.MSLMessageCount
    mov     RxPacketMessageCount, eax

    (point to the first message header in receive buffer)

```

```

ISRProcessMessage:

    (read in message header here)

    mov     eax, MessageHeader.EaxParameter
    mov     ebx, MessageHeader.EbxParameter
    mov     ecx, MessageHeader.EcxParameter
    mov     edx, MessageHeader.EdxParameter
    mov     esi, MessageHeader.EsiParameter
    mov     edi, MessageHeader.EdiParameter

    call    [ReceiveServerCommPointer]           ;inform OS of message
                                                ;(using indirect call)

    ;*****
    ;* the OS may have modified ECX and ESI so save new values      *
    ;*****

    mov     NewEsiParameter, esi
    mov     NewEcxParameter, ecx

    ;*****
    ;* Examine status to determine action for message              *
    ;*****

    cmp     al, 0
    je     ISRCopyMessage

    cmp     al, 1
    je     ISRCopyMessageAndCallBackOS

    cmp     al, 4
    jae    ISRIgnoreMessage

    jmp     ISRHoldOffMessage

ISRProcessNextMessage:

    inc     ReceiveMsgCount                     ;1 more message delivered
    dec     RxPacketMessageCount                ;1 less message to process
    jz     ISRReceiveMessageDone                ;jump if no more

    (point to next message header in receive buffer)

    jmp     ISRProcessMessage                   ;hand next msg to OS

ISRReceiveMessageDone:

    jmp     ISRExit                             ;exit receive handler

```

ISRCopyMessage:

```

    or      ecx, ecx                ;any data to copy?
    jz      ISRProcessNextMessage ;if not, process next message

    (copy message data to OS memory: NewEcx=Size NewEsi=addr)

    jmp     ISRProcessNextMessage ;process next message

```

ISRCopyMessageAndCallBackOS:

```

    or      ecx, ecx                ;any data to copy?
    jz      ISRCallBackOS          ;if not, skip data copy

    (copy message data to OS memory: NewEcx=Size NewEsi=addr)

```

ISRCallBackOS:

```

    mov     eax, MessageHeader.EaxParameter ;original eax parameter
    mov     ebx, MessageHeader.EbxParameter ;original ebx parameter
    mov     ecx, NewEcxParameter          ;use new ecx parameter
    mov     esi, NewEsiParameter          ;use new esi parameter

    call    edx                          ;call to OS

    inc     OSCallBackCount                ;update statistics counter
    jmp     ISRProcessNextMessage         ;process next message

```

ISRIgnoreMessage:

```

    inc     OSRejectMsgCount              ;update statistics counter
    jmp     ISRProcessNextMessage         ;process next message

```

ISRHoldOffMessage:

```

;*****
;* PLEASE NOTE:
;* How the driver handles the HoldState is very adapter specific.
;* The adapter's hardware functions will decide which algorithm the
;* MSL driver will use to handle the holdoff state.
;*
;* Questions must be addressed such as: Can the MSL leave the message
;* packet on the adapter or must the packet be removed immediately upon
;* reception?
;*
;* This example assumes the following:
;*
;* 1. The received packet can not be left on the adapter after
;* reading the message header.
;*
;* 2. The adapter has a receive buffer for more than one
;* maximum size packet.
;*
;*****

```

(copy all remaining parts of the message packet into the receive buffer)

```

    inc     OSHoldMsgCount                ;update statistics counter
    inc     HoldStateFlag                 ;indicate hold state

```

```

mov     ebx, CurrentBufferIndex
mov     CurrentHoldBufferIndex, ebx           ;save index

mov     eax, ebx                             ;update (increment) index
shr     eax, 2                               ;only use 0, 1, & 2 indexes
adc     ebx, 0                               ;if index = 3 then
inc     ebx                                  ;rollover to 0
and     bl, 3
mov     CurrentBufferIndex, ebx             ;save index

mov     eax, CurrentReceiveBufferTable[ebx*4] ;get next buffer addr
xchg   CurrentReceiveBufferAddress, eax     ;set new rx buffer
                                              ;and get held one in
                                              ;into eax

cmp     HoldStateFlag, 2                     ;check if holdstate
je     ISRReceiveMessageDone                ;go on to next receive

;*****
;* Setup for Message redelivery attempts *
;*****

mov     CurrentReceiveHoldBufferAdd, eax     ;set held buffer addr
(point EAX to the message header of the current heldoff message)

mov     CurrentHoldBufferMsgPointer, eax

call    TransmitHoldNotification            ;notify other server
                                              ;of hold state

;*****
;* Schedule callbacks to redeliver message(s) *
;*****

push   OFFSET HoldOffEvent
call   ScheduleSleepAESProcessEvent
add    esp, 1 * 4

mov    edx, OFFSET IntHoldOffEvent
call   ScheduleInterruptTimeCallBack

jmp    ISRReceiveMessageDone

```

```

;*****
;* Acknowledgement Received *
;*****

ISRackReceived:

    cmp     MessageInProgress, TRUE           ;validate send ack
    jne     CheckAdapterStatus              ;poll again on status

;*****
;* Cancel Message TimeOut Sequence *
;*****

    mov     MessageInProgress, FALSE        ;clear flag
    mov     TimeoutEvent.MessageTimeoutTime, 0 ;stop meessage timer

;*****
;* Notify OS of the acknowledgement(s) *
;*****

    mov     ebp, TxPacketMessageCount      ;get # of messages sent
    add     ReceiveAckCount, ebp           ;update statistics counter
    call    [SendServerCommCompletedPointer] ;notify OS of ACKs
                                                ;(use indirect call)

;*****
;* Transmit any queued messages in possible multi-message packet *
;*****

    mov     PacketSizeDriverCanNowHandle, MAX_PACKET_SIZE ;size MSL can now send
    cmp     PacketSizeNowAvailable, MAX_PACKET_SIZE      ;check if OS has messages
    ja     CheckAdapterStatus

(setup to send possible multi-message burst packet)

    mov     TxPacketMessageCount, 0
    call    [GetNextPacketPointer]           ;start BuildSend sequence
                                                ; (using indirect call)
    mov     PacketSizeDriverCanNowHandle, -1 ;semaphore no more sends

    call    TransmitMessagePacket

    inc     TransmitBurstPacketCount        ;update statistics counter
    jmp     CheckAdapterStatus

```



```

;*****
;* Hold Notification Received *
;*****

ISRHoldReceived:

    cmp     MessageInProgress, TRUE           ;validate Hold
    jne     CheckAdapterStatus              ;poll again on status

    mov     eax, ServerCommACKTimeOut       ;extend message timer
    mov     TimeOutEvent.MessageTimeOutTime, ax

    inc     ReceiveHoldCount                ;update statistics counter
    jmp     CheckAdapterStatus              ;poll again on status

;*****
;* Emergency Notification Received *
;*****

ISREmergencyReceived:

    inc     ReceiveEmergencyCount           ;update statistics counter

    cmp     HoldStateFlag, 0                ;check if in holdoff
    jne     ISRPutEmergencyErrorOnHold     ;hold off calling OS

    push    OTHER_SERVER_DEAD_ERROR        ;set error code
    call    ServerCommDriverError          ;notify OS
    add     esp, 1 * 4                       ;cleanup stack
    inc     ServerCommErrorCount           ;update statistics counter
    jmp     CheckAdapterStatus              ;poll again on status

ISRPutEmergencyErrorOnHold:

    mov     EmergencyReceivePending, TRUE   ;set pending flag
    jmp     CheckAdapterStatus              ;poll again on status

```

```

;*****
;* Transmit Complete / Transmit Error *
;*****

ISRTransmitEvent:

    ;;Check for transmit complete. If not transmit complete exit ISR.
    ;;Check for transmit errors and handle (adapter specific).

ISRTransmitComplete:

;*****
;* Cancel Adapter TimeOut Sequence *
;*****

    mov     TransmitInProgress, FALSE        ;clear channel flag
    mov     TimeOutEvent.AdapterTimeOutTime, 0    ;stop timer

;*****
;* Last transmission has completed. Now transmit any waiting sends *
;*****

ISRTransmitWaitingAcks:

    cmp     AckTransmitPending, TRUE          ;any Ack sends waiting?
    jne     ISRTransmitWaitingHolds          ;jump if not

    call    TransmitAcknowledgement          ;update counter
    jmp     CheckAdapterStatus

ISRTransmitWaitingHolds:

    cmp     HoldTransmitPending, TRUE         ;any Hold sends waiting?
    jne     ISRTransmitWaitingMessages       ;jump if not

    call    TransmitHoldNotification
    jmp     CheckAdapterStatus

ISRTransmitWaitingMessages:

    cmp     MessageTransmitPending, TRUE     ;any msg sends waiting?
    jne     CheckAdapterStatus               ;jump if not

    call    TransmitMessagePacket
    jmp     CheckAdapterStatus

;*****
;* DriverISR Exit *
;*****

ISRExit:

    (Place adapter specific code here to enable adapter interrupts)

    ret

DriverISR     endp

```

```

;*****
;* TransmitMessagePacket
;*****

Align 16
TransmitMessagePacket  proc

    cmp     TransmitInProgress, TRUE          ;if a transmit is in progress...
    je     PutMessageTransmitOnHold         ;...jump (can't send msg now)

    (Initiate the transmit of the loaded message packet here)

;*****
;* Begin watching for Adapter Timeout Errors
;*****

    mov     TransmitInProgress, TRUE
    mov     TimeOutEvent.AdapterTimeOutTime, ADAPTER_TIMEOUT_COUNT

;*****
;* Begin watching for Message Timeout Errors
;*****

    mov     MessageInProgress, TRUE
    mov     eax, ServerCommACKTimeOut
    mov     TimeoutEvent.MessageTimeoutTime, ax

    mov     MessageTransmitPending, FALSE
    ret

PutMessageTransmitOnHold:

    mov     MessageTransmitPending, TRUE
    ret

TransmitMessagePacket  endp

```

```
*****
;* TransmitAcknowledgement *
*****

Align 16
TransmitAcknowledgement  proc

    cmp     TransmitInProgress, TRUE           ;if a transmit is in progress...
    je     PutAckTransmitOnHold              ;...jump (can't send Ack now)

    ; (Send an Acknowledgement packet for the number of messages
    ; specified by PacketHeader.MSLMessageCount here)

    mov     TransmitInProgress, TRUE
    mov     TimeoutEvent.AdapterTimeoutTime, ADAPTER_TIMEOUT_COUNT

    mov     HoldTransmitPending, FALSE
    mov     AckTransmitPending, FALSE
    inc     TransmitAckCount
    ret

PutAckTransmitOnHold:

    mov     AckTransmitPending, TRUE
    ret

TransmitAcknowledgement  endp
```

```

;*****
;* TransmitHoldNotification
;*****

Align 16
TransmitHoldNotification      proc

    call    GetCurrentTime      ;Don't hammer the other server
    cmp     eax, LastHoldTransmitTime ;more than once each clock tick
    jne     TransmitHold       ;with holdoff notifications
    ret

TransmitHold:

    cmp     TransmitInProgress, TRUE
    je      PutHoldTransmitOnHold

    mov     LastHoldTransmitTime, eax

    (Send Hold Notification packet to other server)

    mov     TransmitInProgress, TRUE
    mov     TimeOutEvent.AdapterTimeOutTime, ADAPTER_TIMEOUT_COUNT

    mov     HoldTransmitPending, FALSE
    inc     TransmitHoldCount
    ret

PutHoldTransmitOnHold:

    mov     HoldTransmitPending, TRUE
    ret

TransmitHoldNotification      endp

```

```

;*****
;*  DriverTimeout                                     *
;*****
;*
;*  This routine will be executed every tick.  If on entry both the
;*  AdapterTimeoutTime and MessageTimeoutTime counters are zero, the
;*  adapter is idle.  This routine handles timeout events if the adapter
;*  did not complete the transmit or if an acknowledgement is not received.
;*
;*  Assumes:      Interrupts are enabled
;*
;*****
Align 16
DriverTimeout  proc

                CPush
                cli

CheckIfAdapterTimedOut:

                cmp     TimeoutEvent.AdapterTimeoutTime, 0
                je      CheckIfMessageTimedOut

                dec     TimeoutEvent.AdapterTimeoutTime
                jnz     CheckIfMessageTimedOut

                ;*** Adapter Timed Out ***

                inc     AdapterTimedOutCount                ;custom statistics counter
                mov     TransmitInProgress, FALSE
                mov     TimeoutEvent.AdapterTimeoutTime, 0

                mov     eax, HARDWARE_ERROR
                cmp     HoldStateFlag, 0
                je      NotifyError

                mov     HardwareErrorPending, TRUE
                jmp     DriverTimeoutExit

CheckIfMessageTimedOut:

                cmp     TimeoutEvent.MessageTimeoutTime, 0
                je      DriverTimeoutExit

                dec     TimeoutEvent.MessageTimeoutTime
                jnz     DriverTimeoutExit

                ;*** Message Timed Out ***

                inc     MessageTimedOutCount                ;custom statistics counter
                mov     MessageInProgress, FALSE
                mov     TimeoutEvent.MessageTimeoutTime, 0

                mov     eax, TIME_OUT_ERROR
                cmp     HoldStateFlag, 0
                je      NotifyError

                mov     TimeoutErrorPending, TRUE
                jmp     DriverTimeoutExit

```

NotifyError:

```
    push    eax
    call    ServerCommDriverError    ;call OS
    add     esp, 1 * 4                ;clean up stack
    inc     ServerCommErrorCount
```

DriverTimeOutExit:

```
    push    OFFSET TimeOutEvent
    call    ScheduleNoSleepAESProcessEvent
    add     esp, 1 * 4

    CPop
    ret
```

DriverTimeOut endp

```

;*****
;* DriverHoldOff
;*****

Align 16
DriverHoldOff proc near

    CPush
    cli

    cmp     HoldStateFlag, 0
    je     DriverHoldOffExit

    mov     BackOffAmount, 0

AttemptToRedeliverMessage:

    call    HoldOffDeliverMessageToOS
    cmp     HoldStateFlag, 0
    je     CancelDriverIntHoldOff

    inc     BackOffAmount
    mov     eax, BackOffAmount
    mov     HoldOffWaitLoopCount, eax

HoldOffWaitLoop:

    call    CRescheduleLast
    cmp     HoldStateFlag, 0
    je     DriverHoldOffExit

    dec     HoldOffWaitLoopCount
    jnz    HoldOffWaitLoop
    jmp    AttemptToRedeliverMessage

CancelDriverIntHoldOff:

    mov     edx, OFFSET IntHoldOffEvent
    call    CancelInterruptTimeCallBack

DriverHoldOffExit:

    CPop
    ret

DriverHoldOff endp

```



```

;*****
;* DriverIntHoldOff
;*****

Align 16
DriverIntHoldOff      proc      near

    cmp     HoldStateFlag, 0          ;message still on hold?
    je     DriverIntHoldOffExit      ;if not, DriverHoldOff got
                                           ; it delivered already

    call    HoldOffDeliverMessageToOS ;else redeliver message now

    cmp     HoldStateFlag, 0          ;message still on hold?
    je     DriverIntHoldOffExit      ;if not, we got it delivered

    mov     edx, OFFSET IntHoldOffEvent ;otherwise, reschedule callback
    call    ScheduleInterruptTimeCallBack ; to this routine

DriverIntHoldOffExit:

    ret                                     ;exit if done

DriverIntHoldOff      endp

```

```

;*****
;* HoldOffDeliverMessageToOS
;*****

Align 16
HoldOffDeliverMessageToOS      proc

HoldProcessMessage:

    mov     ebp, CurrentHoldBufferMsgPointer      ;get message pointer

    mov     eax, [ebp].EaxParameter
    mov     ebx, [ebp].EbxParameter
    mov     ecx, [ebp].EcxParameter
    mov     edx, [ebp].EdxParameter
    mov     esi, [ebp].EsiParameter
    mov     edi, [ebp].EdiParameter

    call    [ReceiveServerCommPointer]

    mov     HoldNewEsiParameter, esi
    mov     HoldNewEcxParameter, ecx

;*****
;* Examine status to determine action for message
;*****

    cmp     al, 0
    je      HoldCopyMessage

    cmp     al, 1
    je      HoldCopyMessageAndCallBackOS

    cmp     al, 4
    jae     HoldIgnoreMessage

    jmp     HoldOffMessage

HoldProcessNextMessage:

    inc     ReceiveMsgCount                      ;1 more message delivered
    dec     RxPacketMessageCount                ;1 less message to process
    jz      HoldProcessNextHoldPacket           ;jump if no more

    (point CurrentHoldBufferMsgPointer to next message header in hold buffer)

    jmp     HoldProcessMessage                  ;hand next msg to OS

HoldProcessNextHoldPacket:

    call    HoldCheckIfErrorPending

    cmp     HoldStateFlag, 0
    je      HoldOffDeliverMessageToOSExit

    dec     HoldStateFlag
    jz      HoldOffDeliverMessageToOSExit

    call    TransmitAcknowledgement              ;ack 2nd message packet

    mov     ebx, CurrentHoldBufferIndex          ;update index
    mov     eax, CurrentReceiveBufferTable[ebx*4] ;get new buffer ptr
    mov     CurrentReceiveHoldBufferAdd, eax    ;set new buffer ptr

    (point CurrentHoldBufferMsgPointer to first message header in hold buffer)

    jmp     HoldProcessMessage

```

HoldCopyMessage:

```

mov     ebp, CurrentHoldBufferMsgPointer    ;get message pointer
or      ecx, ecx                            ;any data to copy?
jz      HoldProcessNextMessage             ;if not, process next message

(copy message data from Hold Buffer to OS memory)
(      HoldNewEcxParameter = size          )
(      HoldNewEsiParameter = destination   )

jmp     HoldProcessNextMessage

```

HoldCopyMessageAndCallBackOS:

```

mov     ebp, CurrentHoldBufferMsgPointer    ;get message pointer
or      ecx, ecx                            ;any data to copy?
jz      HoldCallBackOS                     ;if not, skip data copy

(copy message data from Hold Buffer to OS memory)
(      HoldNewEcxParameter = size          )
(      HoldNewEsiParameter = destination   )

```

HoldCallBackOS:

```

mov     eax, [ebp].EaxParameter
mov     ebx, [ebp].EbxParameter
mov     ecx, HoldNewEcxParameter
mov     esi, HoldNewEsiParameter

call    edx                                ;call to OS

inc     OSCallBackCount                    ;update statistics counter
jmp     HoldProcessNextMessage

```

HoldIgnoreMessage:

```

inc     OSRejectMsgCount                   ;update counter
dec     ReceiveMsgCount                    ;fix for later ???
jmp     HoldProcessNextMessage

```

HoldOffMessage:

```

inc     OSHoldMsgCount
inc     HoldStateFlag
call    TransmitHoldNotification           ;transmit hold notification

```

HoldOffDeliverMessageToOSExit

```
ret
```

```
HoldOffDeliverMessageToOS    endp
```

```

;*****
;* HoldCheckIfErrorPending
;*****

Align 16
HoldCheckIfErrorPending  proc

    cmp     EmergencyReceivePending, TRUE
    mov     eax, OTHER_SERVER_DEAD_ERROR
    je     HoldNotifyError

    cmp     HardwareErrorPending, TRUE
    mov     eax, HARDWARE_ERROR
    je     HoldNotifyError

    cmp     TimeOutErrorPending, TRUE
    mov     eax, TIME_OUT_ERROR
    je     HoldNotifyError

    ret

HoldNotifyError:

    push    eax
    call    ServerCommDriverError
    add     esp, 1 * 4
    inc     ServerCommErrorCount

;*****
;* Reset "everything" after error
;*****

    mov     EmergencyReceivePending, FALSE
    mov     HardwareErrorPending, FALSE
    mov     TimeOutErrorPending, FALSE

    mov     TransmitInProgress, FALSE
    mov     MessageInProgress, FALSE

    mov     TimeOutEvent.MessageTimeOutTime, 0
    mov     TimeOutEvent.AdapterTimeOutTime, 0

    mov     HoldStateFlag, 0          ;drop possible 2nd hold packet

    ret

HoldCheckIfErrorPending  endp

```

```

;*****
;* DriverRemove
;*****

Align 4
DriverRemove    proc

    CPush
    pushfd
    cli

;*****
;* Unhook from Interrupt vector
;*****

    push    OFFSET DriverISR
    movzx   eax, BYTE PTR DriverConfiguration.CInterrupt0
    push    eax
    call    ClearHardwareInterrupt
    add     esp, 2 * 4

;*****
;* See if we are currently in a Holdoff state
;*****

    cmp     HoldStateFlag, 0
    je      CancelCallBackEvents

;*****
;* Wait until the holdoff state is finished to cancel
;*****

TryAgain:

    call    CRescheduleLast
    cmp     HoldStateFlag, 0
    jne     TryAgain

CancelCallBackEvents:

    mov     edx, OFFSET IntHoldOffEvent
    call    CancelInterruptTimeCallBack

    push    OFFSET HoldOffEvent
    call    CancelSleepAESProcessEvent
    add     esp, 1*4

    push    OFFSET TimeOutEvent
    call    CancelNoSleepAESProcessEvent
    add     esp, 1*4

;*****
;* Deregister driver from OS
;*****

    push    MSLDriverResourceTag                ;pass Resource tag
    call    DeRegisterServerCommDriver          ;remove the driver.
    add     esp, 1*4

```

```

;*****
;* Deregister hardware options from OS
;*****

push    OFFSET DriverConfiguration
call    DeRegisterHardwareOptions
add     esp, 1*4

popfd
CPop
ret

DriverRemove    endp

OSCODE ends

end
```