NESL Specification: 16-Bit DOS Client Programmer's Interface

NESL Specification Version 1
Document Version 1.04
Part Number: 107–000066–001
6 May 1994

Disclaimer

Novell, Inc. makes no representations or warranties with respect to the contents or use of this manual, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Novell, Inc.
122 East 1700 South
Provo, Utah 84606

Trademarks

Novell has made every effort to supply trademark information about company names, products, and services mentioned in this document. Trademarks indicated below were derived from various sources.

Novell and NetWare are registered trademarks of Novell, Inc.

MLID and Multiple Link Interface Driver are trademarks of Novell, Inc.

Specification Changes

The following change has been made to this specification since document version 1.03 (2 March 1994):

The ASCIIZ string “EVENTS” has been changed to “NESL_EVENTS”.

Table Of Contents

List of Figures

List of Tables

Introduction

The NetWare Event Service Layer (NESL) for 16-Bit DOS allows modules in a system to receive notification about certain events that can occur in the other modules, such as events generated by PCMCIA or APM. However, the NESL is intended to be generic and is not limited to these events.

The NESL works as a DOS TSR (terminate and stay resident) program that provides a set of services any module in the system can use. These services provide a mechanism for generating an event and/or receiving notification when an event occurs. Modules that generate events are referred to as producers; those that receive notification of events are consumers. A module can be both a producer and a consumer.

The event notification is based on event classes. Event classes are defined for a specific group or category of events. A module that produces an event registers as a producer of the event class that the event belongs to. There can be more than one producer of an event class. The NESL keeps track of producers of event classes and maintains a list of consumers for each event class.

When an event occurs, consumers of the event class that the event occurred within are notified. The NESL is not aware of individual events and does not attempt to keep track of them. However, the module that generated the event passes detailed information about the particular event to consumers at notification time. This allows modules to control the level of sensitivity they have to an event notification. For example, if a particular module is only interested in knowing that service has been suspended and does not care about the cause, it can take action every time it receives notification for the “Service Suspend” event class. On the other hand, if a module is only interested in “MLID Card Removal” events within the “Service Suspend” event class, it can filter on the information provided with the notification for the “Service Suspend” event class and take action only if the notification is a result of an “MLID Card Removal” event.

Example: An MLID for a PCMICA card registers as a client of Card Services during load time. It also registers with the NESL as a producer of the “Service Suspend” and “Service Resume” event classes. When the MLID receives and successfully processes a card insertion event, it calls the NESL’s Service Entry Point to generate an “MLID Card Insertion Complete” event. After the MLID receives and processes a card removal

event from Card Services, it calls the NESL's Service Entry Point to generate an "MLID Card Removal" event. Protocol stacks or other modules that registered with the NESL as consumers of the "Service Suspend" and "Service Resume" event classes receive notification when the events takes place.

The NetWare Event Service Layer

The NetWare Event Service Layer's (NESL) links producers of events with the consumers of those events. The NESL provides the following services:

- Get NESL Config Pointer
- Register Event Producer
- Deregister Event Producer
- Event Notification
- Register Event Consumer
- Deregister Event Consumer

Registering and Deregistering Event Producers

Event producers use *RegisterEventProducer* to register with the NESL as a producer of an event class. Once it registers, the event producer calls *EventNotification* to notify event consumers when an event takes place.

Note Event producers can also register as event consumers.

When an event producer no longer provides events within an event class, it calls *DeRegisterEventProducer* for that class. For example, an event-providing module that is unloading its clean-up function must first call *DeRegisterEventProducer* for each event class it has added. The module could then complete its unloading process.

Registering and Deregistering Event Consumers

Event consumers must register with the NESL in order to receive notification when an event occurs. These modules call *RegisterEventConsumer* for each event class they wish to be notified of.

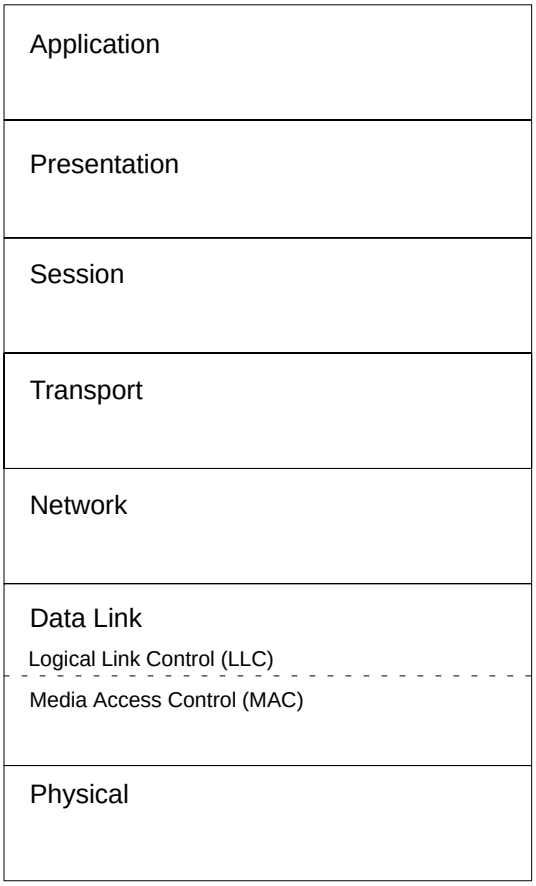
When an event consumer no longer requires event notification, or before it unloads, it must deregister by calling *DeRegisterEventConsumer* for each class it registered for.

Signaling Events

The NESL maintains a consumer list for each event class. When an event producer calls the NESL to signal that an event has occurred within a class, the NESL notifies all modules on the consumer list. The order in which the NESL calls the consumers depends upon the OSI layer to which the consumer belongs and upon the direction in which the event notification must take place in the OSI model. (Figure 1 illustrates the OSI model.) The NESL tracks the responses from each module in

the list and returns a collective response back to the event producer.

Figure 1
The OSI Model



The NESL Programmer's Interface

This section describes the interface that allows other modules to access NESL services.

Locating the NESL

To locate the NESL, a module must scan the DOS interrupt 2F multiplex IDs between C0h and FFh for the NESL signature string. The module must scan because the NESL uses the interrupt 2F ID dynamically. When the module calls interrupt 2F with the ID of the NESL's location, the interrupt returns with DX:BX containing the address of the NESL's Service Entry Point and with ES:SI pointing to the NESL's signature. This signature is defined as the ASCIIZ string 'NESL_EVENTS'. To use the NESL's services, the module must call the service entry point with registers set as outlined by the particular function.

Sample Code for Locating the NESL

The following is sample code for finding the Event Service Layer:

```
NESLSignature      db  'NESL_EVENTS', 0 ; NESL Signature

MULTIPLEX_INT      equ     2Fh         ; Multiplex Interrupt.
MUL_CHCKSLT_AVAIL  equ     0C000h      ; User defined slot 0 and
                                       ; Function 0
SLOT_IN_USE        equ     0FFh        ; Slot is in use.
ENTER_DOS          equ     21h         ; DOS Interrupt.
GET_INT_VEC        equ     35h         ; Get Interrupt Vector.
FAILED             equ     8005h       ; Failed error code.


NESL   proc    near
     push    si
     push    di
     push    bp
     push    ds
     push    es
     mov     ax, (GET_INT_VEC SHL 8) OR MULTIPLEX_INT
     int     ENTER_DOS                 ; Let DOS Get vector for us.

     cmp word ptr es:[bx], 0           ; Check current Vector.
     je   NESLNotPresentExit           ; Has it been hooked?

;_____________________________________________________________________
;    Scan through the 2Fh slots and look for loaded NESL.
;_____________________________________________________________________

cld
mov     ax, MUL_CHCKSLT_AVAIL      ; Start after DOS reserved

LookForNESLLoop:
     push    ax                    ; AH = Slot # and
                                   ; AL= Function 0
     push    ds                    ; DS = CGroup
     int     MULTIPLEX_INT         ; Hit the Multiplex Int functions
     cmp     al, SLOT_IN_USE       ; Slot used?
     pop     ds                    ; DS = CGroup
     pop     ax                    ; AH = Slot # and
                                   ; AL = Function 0
     je      Int2FSlotUsed         ; Was slot in use?

;_____________________________________________________________________
;   Next Slot Please
;_____________________________________________________________________

GetNextSlot:
     inc     ah                    ; AH = Slot # to check next
     jnz     LookForNESLLoop       ; if we wrap after FF then we
                                   ; are done.
     jmp short NESLNotPresent Exit ; NESL not present.

;_____________________________________________________________________
;   If the slot is used, is it the NESL?
;_____________________________________________________________________
```

```
Int2FSlotUsed:
      mov     di, si              ; ES:DI –> signature string
      mov     si, offset DGROUP:NESLSignature
      mov     cx, 3               ; Check 6 bytes.
rep cmpsw
       jnz GetNextSlot            ; Not it, keep looking

;—————————————————————————————————————————
; Found the NESL. DX:BX contains address of the NESL's Entry Point.
;—————————————————————————————————————————

xor           ax, ax              ; Successful return code.

FindNESLExit:
      pop     es
      pop     ds
      pop     bp
      pop     di
      pop     si
      ret

NESLNotPresentExit:
      mov     ax, FAILED          ; Error return code.
      or      ax, ax
      jmp     FindNESLExit

FindNESL   endp
```

NESL Services

This section describes the services provided by the NESL.

GetNESLConfigPointer

Description

Returns a pointer to the NESL's configuration table.

Entry State

BX

0000h

Return State

AX

Set to a completion code.

ES:SI

Pointer to the NESL's configuration table if call is successful.

Preserved

DS, BP, SS, SP.

Completion Codes (AX)

0000h *Successful*

The service is available and ES:SI contains a valid pointer to the NESL's configuration table.

8008h *Bad Command*

The service is not available.

RegisterEventProducer

Description

Allows a module to register as a producer of a particular event class.

Entry State

BX

0001h

ES:SI

Pointer to a Producer Event Control Block (PECB) with the following fields set to:

PECB_ClassName Pointer to the ASCIIZ string which identifies the class of events.

PECB_Flags *bit 0*

Set to 0 if consumers are to be called top\down. (This is based on the value in the Consumer Event Control Block's [CECB] *OSILevel* field; consumers are called from 7 to 1).

Set to 1 if consumers are to be called bottom\up. (This is based on the value in the CECB's *OSILevel* field; consumers are called from 1 to 7).

bits 1–31

Reserved. Set to zero.

Return State

AX

Set to a completion code.

ES:SI

Pointer to the PECB that was passed into the function.

Preserved

DS, BP, SS, SP.

Completion Codes (AX)

0000h *Successful*

The calling module was successfully registered as a producer of the given event class.

8005h *Fail*

The calling module is already registered as a producer of the given event class.

8008h *Bad Command*

The service is not available.

Remarks

RegisterEventProducer registers a module as a producer of a particular event class. This routine contains, and returns, a pointer to a Producer Event Control Block (PECB). Although the event producer provides the memory for the PECB, the NESL owns this memory until the event producer deregisters by calling *DeRegisterEventProducer*.

RegisterEventProducer *continued*

Once the application registers, the only PECB field it can change is the *PECB_DataPtr* field. The application might need to modify this field when it calls *EventNotification*. Until it deregisters, the application must not change any other PECB field.

The calling module sets the PECB *EventClassName* field to point to an ASCIIZ string identifying the event class. Refer to the "Defined Classes and Events" section for a list of class names Novell has defined. The NESL initializes all other fields.

See Also

DeRegisterEventProducer
EventNotification

DeRegisterEventProducer

Description

Allows a module to deregister as a producer of an event class that it previously registered.

Entry State

BX

0002h

ES:SI

Pointer to the Producer Event Control Block (PECB) that was passed into *RegisterEventProducer* for a given event class.

Return State

AX

Set to a completion code.

ES:SI

Pointer to the PECB was passed into the function.

Preserved

DS, BP, SS, SP.

Completion Codes (AX)

0000h *Successful*

The caller was successfully deregistered as a producer of the given event class.

8002h *Bad Parameter*

The caller was not registered with the NESL as a producer of the given event class.

8008h *Bad Command*

The service is not available.

See Also

RegisterEventProducer

EventNotification

Description

Allows a module to signal that an event has just occurred in a given class.

Entry State

BX

0003h

ES:SI

Pointer to the Producer Event Control Block (PECB) that was passed into the *RegisterEventProducer* for the given event class. The following PECB field must be set to:

PECB_DataPtr	Pointer to an Event Parameter Block (EPB) filled out as shown below. **Note:** The calling module must not modify any of the other fields in the PECB until the module calls *DeRegisterEventProducer* for the given event class.

The fields of the EPB are set to the following:

EPB_ClassName	Pointer to an ASCIIZ string that identifies the event class.
EPB_EventName	Pointer to an ASCIIZ string that identifies the event within the class.
EPB_ModuleName	Pointer to an ASCIIZ string that identifies the module that generated the event.
EPB_DataPtr0	Pointer to a block of information about the event. This field is event dependent and is defined by the event. If the event has no additional information to pass, this field will be NULL.
EPB_DataPtr1	Pointer to a block of information about the event. This field is event dependent and is defined by the event. If the event has no additional information to pass, this field will be NULL.

Return State

AX

Set to a completion code.

ES:SI

Pointer to the PECB that was passed into the function.

Preserved

DS, BP, SS, SP.

Completion Codes (AX)

0000h *Successful*

The notification was successful.

8005h *Fail*

The notification was generated by a request that was denied by one or more consumers.

8008h *Bad Command*

The service is not available.

Remarks

Refer to the "Defined Classes and Events" section for a list of class and event names that Novell has defined.

When a module calls the notification procedure for each event consumer of the class, it passes a pointer to an EPB as a parameter. This block is considered to be read-only and the event consumer must not modify it.

See Also

Consumer Notification Procedure
DeRegisterEventProducer
RegisterEventProducer

RegisterEventConsumer

Description

Allows a module to register as a consumer of a particular event class.

Entry State

BX

0004h

ES:SI

Pointer to a Consumer Event Control Block (CECB) with the following fields set to:

CECB_ClassName	Pointer to an ASCIIZ string which identifies the event class.
CECB_NotifProc	The address of a procedure to call when an event in the given class occurs.
CECB_OSILevel	The level of the OSI layer which the consumer is located in. This value takes the form of 0xLX, where L is the number (1–7) corresponding to the application's OSI layer and X is the order relative to other modules also registered on that layer. The relative ordering is useful when several system components on the same layer require a certain processing order during the event.

Return State

AX

Set to a completion code.

ES:SI

Pointer to the CECB that was passed into the function.

Preserved

DS, BP, SS, and SP.

Completion Codes (AX)

0000h *Successful*

The caller was registered successfully as a consumer of the given event class.

8005h *Fail*

The caller has already registered as a consumer of this event class.

8008h *Bad Command*

The service is not available.

Remarks

RegisterEventConsumer registers a module as a consumer of a particular event class. This routine contains, and returns, a pointer to a Consumer Event Control Block (CECB). Although the event producer provides the memory for the CECB, the NESL owns this memory until the event consumer deregisters

by calling *DeRegisterEventConsumer*. Once the application registers, and until it deregisters, it must not change any of the CECB fields.

See Also DeRegisterEventConsumer

DeRegisterEventConsumer

Description

Allows a module to deregister as a consumer of an event class.

Entry State

BX

0005h

ES:SI

Points to the Consumer Event Control Block (CECB) that was passed into *RegisterEventConsumer* for the given event class.

Return State

AX

Set to a completion code.

ES:SI

Pointer to the CECB that was passed into the function.

Preserved

DS, BP, SS, SP.

Completion Codes (AX)

0000h *Successful*

The caller has been deregistered as a consumer of the given event class.

8002h *Bad Parameter*

The caller was not registered with the NESL as a consumer of the given event class.

8008h *Bad Command*

The service is not available.

See Also

DeRegisterEventConsumer

Consumer Notification Procedure

Description

Called by the NESL to notify the consumer when an event has occurred with an event class for which it has registered.

Entry State

ES:SI
Pointer to an Event Parameter Block.

Return State

AX
Set to a completion code.

ES:SI
Pointer to the Event Parameter Block (EPB) passed into the notification procedure.

Preserved
DS, BP, SS, SP, DI.

Completion Codes (AX)

Request type events

0000h *Successful*
Request granted.

8005h *Fail*
Request denied.

All Other types of events

0000h *Successful*

Remarks

When the NESL calls the notification procedure for each consumer of a particular event class, it passes the consumer a pointer to the EPB. This block is considered to be read-only, and the event consumer must not modify it.

The NESL obtains the address of the notification procedure from the CECB *CECB_NotifiProc* field that was passed to *RegisterEventConsumer*.

See Also

EventNotification
RegisterEventConsumer

Data Definitions

Signatures

NESLSignature

NESLSignature is defined as the ASCIIZ string 'NESL_EVENTS'

Tables

Configuration Table

The configuration table is defined below.

```
NESLConfigTable   label     byte
    NESL_Cfg_MajVer       dw  1
    NESL_Cfg_MinVer       dw  0
    NESL_Cfg_ModLName     dd  ?
    NESL_Cfg_ModSName     dd  ?
    NESL_Cfg_ModMajVer    dw  1
    NESL_Cfg_ModMinVer    dw  0
```

Table 1 The Configuration Table Fields

The Configuration Table Fields	
Field Name	**Description**
NESL_Cfg_MajVer	This field contains the major version number of the configuration table. The current major version is 1. This field will change whenever major changes have been made to the format of the configuration table or field definitions. The field will also change whenever the *NESL_CFG_MinVer* field exceeds its maximum value. Novell controls this field.
NESL_Cfg_MinVer	This field contains the minor version number of the configuration table. The current minor version is 0. This field will change whenever minor changes are made to the format of the configuration table or field definitions. It will also be reset to zero whenever the *NESL_CFG_MajVer* changes. The maximum value for this field is 99. Novell controls this field.
NESL_Cfg_ModLName	This field contains a pointer to the NESL's long name. The name is defined as the ASCIIZ string 'NetWare Event Service Layer for 16-Bit DOS'.
NESL_Cfg_ModSName	This field has a pointer to the NESL's short name. The name is defined as the ASCIIZ string 'NESL'.

The Configuration Table Fields *(continued)*

Field Name	Description
NESL_Cfg_ModMajVer	This field contains the major version number of the NESL. The initial major version is 1. This field will change whenever major changes have been made to the NESL or whenever the *NESL_CFG_ModMinVer* field exceeds its maximum value. Novell controls this field.
NESL_Cfg_ModMinVer	This field contains the minor version number of the NESL. The current minor version is 0. This field will change whenever minor changes are made to the NESL. It will also be reset to zero whenever the *NESL_CFG_ModMajVer* changes. The maximum value for this field is 99. Novell controls this field.

Structures

Producer Event Control Block (PECB)

```
PECB    struc
    PECB_MajVer          dw  1
    PECB_MinVer          dw  0
    PECB_NextProducer    dd  0
    PECB_ClassName       dd  0
    PECB_ConsumerList    dd  0
    PECB_DataPtr         dd  0
    PECB_Flags           dd  0
    PECB_Reserved        db  8 dup (0)
PECB    ends
```

Table 2
The PECB Fields

The PECB Fields	
Field Name	**Description**
PECB_MajVer	This field contains the major version number of the PECB. The current major version is 1. This field will change whenever major changes have been made to the format of the configuration table or field definitions. This field will also change whenever the *PECB_MinVer* field exceeds its maximum value. Novell controls this field.
PECB_MinVer	This field contains the minor version number of the PECB. The current minor version is 0. This field will change whenever minor changes are made to the format of the configuration table or field definitions. It will also be reset to zero whenever the *PECB_MajVer* changes. The maximum value for this field is 99. Novell controls this field.
PECB_NextProducer	This field contains a pointer to the next Producer Event Control Block in the list. This field will be a NULL if it is the last PECB in the list.
PECB_ClassName	This field contains a pointer to an ASCIIZ string which identifies the event class.
PECB_ConsumerList	This field contains a pointer to a list of consumers for the event class. This field will be NULL if there are no consumers of the event class.
PECB_DataPtr	This field contains a pointer used to pass in additional information. This field is generally used during event notification.

The PECB Fields *(continued)*

Field Name	Description
PECB_Flags	bit 0 **Set to 0** if consumers are to be called top\down. Based on the value in the CECB's *OSILevel* field, they are called from 7 to 1. **Set to 1** if consumers are to be called b ottom\up. Based on the value in the CECB's *OSILevel* field, consumers are called from 1 to 7). bits 1–31 **Reserved**. Set to zero.
PECB_Reserved	Reserved. Set to zero.

Although the event producer provides the memory for the PECB, the NESL owns this memory until the event producer deregisters by calling *DeRegisterEventProducer*. Once the application registers, the only PECB field it can change is the *PECB_DataPtr* field. The application might need to modify this field when it calls *EventNotification*. Until it deregisters, the application must not change any other PECB field.

Consumer Event Control Block (CECB)

```
CECB   Struc
    CECB_MajVer          dw  1
    CECB_MinVer          dw  0
    CECB_NextConsumer    dd  0
    CECB_ClassName       dd  0
    CECB_NotifiProc      dd  0
    CECB_OSILevel        dw  0
    CECB_Reserved        db  14 dup (0)
CECB   ends
```

Table 3
The CECB Fields

The CECB Fields	
Field Name	**Description**
CECB_MajVer	This field has the major version number of the CECB. The current major version is 1. This field will change whenever major changes have been made to the format of the configuration table or field definitions. The field will also change whenever the *CECB_MinVer* field exceeds its maximum value. Novell controls this field.
CECB_MinVer	This field has the minor version number of the CECB. The current minor version is 0. This field will change whenever minor changes are made to the format of the configuration table or field definitions. It will also be reset to zero whenever the *CECB_MajVer* changes. The maximum value for this field is 99. Novell controls this field.
CECB_NextConsumer	This field contains a pointer to the next consumer. This field will be a NULL if it is the last CECB in the list.
CECB_ClassName	This field contains a pointer to an ASCIIZ string which identifies the event class.
CECB_NotifiProc	This field contains the address of a procedure to call when an event occurs.
CECB_OSILevel	The application uses this field at registration time to specify which order the consumers are to be called in when an event occurs. This field's value takes the form of 0xLX, where L is the number (1–7) corresponding to the applications OSI Layer and X is the relative order with other modules also registered on that layer. The relative ordering is useful when several system components on the same level require a certain processing order for the event.
CECB_Reserved	Reserved. Set to zero.

Although the event consumer provides the memory for the CECB, the NESL owns this memory until the event consumer deregisters by calling *DeRegisterEventConsumer*. Once the

application registers. and until it deregisters, it must not modify any CECB field.

Event Parameter Block (EPB)

```
EPB     struc
    EPB_MajVer          dw  1
    EPB_MinVer          dw  0
    EPB_ClassName       dd  ?
    EPB_EventName       dd  ?
    EPB_ModuleName      dd  ?
    EPB_DataPtr0        dd  ?
    EPB_DataPtr1        dd  ?
    EPB_Reserved        db  8 dup (0)
EPB     ends
```

Table 4
The EPB Fields

The EPB Fields	
Field Name	**Description**
EPB_MajVer	This field has the major version number of the EPB. The current major version is 1. This field will change whenever major changes have been made to the format of the configuration table or field definitions. This field will also change whenever the *EPB_MinVer* field exceeds its maximum value. Novell controls this field.
EPB_MinVer	This field contains the minor version number of the EPB. The current minor version is 0. This field will change whenever minor changes are made to the format of the configuration table or field definitions. It will also be reset to zero whenever the *EPB_MajVer changes*. The maximum value for this field is 99. Novell controls this field.
EPB_ClassName	This field has a pointer to ASCIIZ string that identifies the event class.
EPB_EventName	This field has a pointer to an ASCIIZ string that identifies the event within the class.
EPB_ModuleName	This field has a pointer to an ASCIIZ string that identifies the module that produced the event.

The EPB Fields *(continued)*

Field Name	Description
EPB_DataPtr0	This field has a pointer to a block of information about the event. This field is event dependent and will be defined by the event. If the event has no additional information to pass, this field will be NULL.
EPB_DataPtr1	This field has a pointer to a block of information about the event. This field is event dependent and will be defined by the event. If the event has no additional information to pass, this field will be NULL.
EPB_Reserved	Reserved. Set to zero.

The event producer provides the memory for the EPB.

Classes and Events

Defining Classes and Events

Anyone can define an event class as well as the events within any class by defining unique names for them. These names must be ASCIIZ strings. The class definition must include a description of the class and the direction in which the event consumers will be called: an event consumer can be called either from the top of the OSI model down or from the bottom of the OSI model up (See Figure 1).

Events that are added to existing classes must fall within the definition of that class.

Note Contact Novell Labs to register new event classes and event names.

Defined Classes and Events

Event classes and specific events within the class are identified with ASCIIZ strings. We have defined four classes of events along with some specific events within each class.

We have defined the following list of event classes.

- Service Suspend
- Service Resume
- Service/Status Change
- Suspend Request

These classes and their events are described on the following pages.

Service Suspend

The Service Suspend event class contains events that suspends a service. This class is called from the top of the OSI model down.

Table 5
Events in the Service Suspend Class

Events in the Service Suspend Class	
Event Name	**Description**
MLID Cable Disconnect	This event indicates that the cable has been disconnected from a given NIC. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Card Removal	This event is triggered by the hardware and indicates that the PC Card has been removed. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block. Even though this event puts the MLID into shutdown mode, it does not generate a shutdown event.
MLID Hardware Failure	This event indicates that a serious hardware failure has occurred with the NIC. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Not In Range	This wireless event indicates that there is no access point in range. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Shutdown	This event is triggered through the MLID control services and indicates that the MLID was shutdown. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Media Access Denied	This event indicates that access to the physical medium was either denied or unsuccessful. A pointer to the MLID's configuration table is passed in the Event Parameter Block *EPB_DataPrt0* field.

Service Resume

The Service Resume event class contains events that indicate either the availability of a new service or the restoration of a previously available service. This class is called from the bottom of the OSI model up.

Table 6
Events in the Service Resume Class

Events in the Service Resume Class	
Event Name	**Description**
MLID Cable Reconnect	This event indicates that the cable has been reconnected to a given NIC. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Card Insertion Complete	This event is triggered by the hardware and indicates that a PC card has been inserted in a socket and that the MLID and LAN adapter are fully functional. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block. This event does not trigger a reset event.
MLID In Range	This wireless event indicates that an access point is in range again. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Reset	This event is triggered by the MLID control services and indicates that an MLID was just reset. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.

Service/Status Change

The Service/Status event class contains events that signal a change in either the status or the current level of service. This class is called from the top of the OSI model down.

Table 7
Events in the Service/ Status Class

Events in the Service/Status Class	
Event Name	**Description**
MLID Access Point Change	This event indicates that a station has moved from one access point's range to another and that the new access point will start serving the station. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.
MLID Speed Change	This event indicates that there has been a change in the communication speed. For example, in the wireless environment this could be caused by the radio link due to a change in the quality of the signal. A pointer to the MLID's configuration table is passed in the *EPB_DataPtr0* field of the Event Parameter Block.

Suspend Request

The Suspend Request class contains events that request permission to suspend the services. This class is called from the bottom of the OSI model up.

Currently no events have been defined for this class.

❑