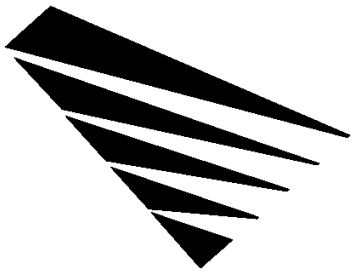

Appendix A
NetWare Event Service Layer

The NetWare Event Service Layer (NESL) handles event registry and notification. NESL is provided as part of the event service API in NIOS. The NESL component is portable. In fact it can be used independent of NIOS on other platforms such as the NetWare OS.

Event Registery and Notification

NESL is designed around the concept of consumers and producers. Generally, a producer will produce an event and a consumer will consume the event.

For a given event type, there can be many producers and many consumers simultaneously. A Client module must register as a producer of an event type in order to produce that event. Likewise, a module must register as a consumer of an event type in order to consume the event. A module can register as both a producer and consumer of the same event type.

When a producer registers to produce an event, it specifies whether it is the sole producer of the event, and whether the event is consumable or broadcast. If the event is broadcast, notification is sent to every registered consumer. If consumable, notification of the event is sent to every consumer unless one of them consumes the event..

If a consumer chooses to consume an event, it will notify the producer that the event is consumed, and event notification will end.

When a producer or consumer is removed from the system, it must deregister all producer/consumer events it has registered.

Note: NIOS is currently an event driven subsystem. Tasks should be designed to run to completion. If producer and consumer routines are running on asynchronous events (e.g. IPX packet interrupts) then the routines must be re-entrant. **NESLProduceEvent** will not protect the consumer routine from being reentered.

The NESL maintains a consumer list for each event type. When a producer calls the NESL to signal that an event has occurred within a class, the NESL notifies everyone in the consumer list. The order that the consumers are called depends on level of the OSI model the consumer belongs to and the calling direction defined by the event class.

NESL APIs

Following is a brief description of each of the functions that are available for the NESL. The remainder of this chapter contains full details of these functions.

NESLDeRegisterConsumer Deregisters the specifed consumer of an event.

NESLDeRegisterProducer Deregisters the specified producer.

NESLEnumerateEvents Enumerates the registered events.

NESLProduceEvent Notifies registered consumers that an event has occurred.

NESLRegisterConsumer Registers a consumer of an event.

NESLRegisterProducer Registers an event producer.

NESLScanConsumersByName Enumerates the registered consumer for the specified event.

NESLScanProducersByName Enumerates the registered producers for the specified event.

NESLDeRegisterConsumer

Description

Deregisters a consumer of the specified event.

Syntax

```
UINT32
NESLDeRegisterConsumer(
        NESL_ECB      *Consumer);
```

Parameters

Consumer The NESL_ECB passed to **NESLRegisterConsumer**

Return values

NESL_OK Deregistry succeeded.

NESL_EVENT_NOT_REGISTERED
The specified NESL_ECB is not registered.
NESL_CONSUMER_NOT_FOUND
The consumer is NULL or cannot be located.

Remarks

Called from foreground with interrupts enabled.

See also

NESLRegisterConsumer
NESLRegisterProducer
NESLDeRegisterProducer
NESLProduceEvent
NESLEnumerateEvents
NESLScanProducersByName
NESLScanConsumersByName

NESLDeRegisterProducer

Description

Deregisters the producer. If the producer is the last producer of its type, this routine will place any remaining consumers of the event onto an orphaned consumer's list.

Syntax

```
UINT32
NESLDeRegisterProducer(
        NESL_ECB      *Producer);
```

Parameters

Necb The NESL_ECB passed to **NESLRegisterProducer**.

Return values

NESL_OK Deregistry succeeded.

NESL_EVENT_NOT_REGISTERED
The event cannot be located.

NESL_PRODUCER_NOT_FOUND
The producer is NULL or couldn't be located.

Remarks

Called from foreground with interrupts enabled.

See also

NESLRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLProduceEvent
NESLEnumerateEvents
NESLScanProducersByName
NESLScanConsumersByName

NESLEnumerateEvents

Description

Returns a copy of the specified *EventName* to the caller's buffer.

Syntax

```
UINT32
NESLEnumerateEvents(
        void            **ContextHandle,
        MEON_STING      *EventName );
```

Parameters

ContextHandle Context of the previous call to **NESLEnumerateEvents**. Equals 0 if this is the first call.

EventName The destination for a copy of the event name. This destination MUST already have allocated memory associated with it. The maximum length for an event name is MAX_EVENT_NAME (81 bytes).

Return values

NESL_EVENT_IS_CONSUMABLE
The returned event is consumable.

NESL_EVENT_IS_NOT_CONSUMABLE
The returned event is not consumable.

NESL_INVALID_DESTINATION
EventName is a NULL pointer.

NESL_NO_MORE_EVENTS
There are no more events to retur

Remarks

If *ContextHandle* is NULL, the first registered event is used. Otherwise, **ContextHandle* specifies the previous context and the next event is used.

Note: Enumerating events must be done from a single thread or context of execution. Because any number of consumers or producers may register or deregister events when the foreground is relinquished, relinquishing control will invalidate the ContextHandle.

The caller MUST provide a buffer with *MAX_EVENT_NAME* space already allocated.
This call will NOT enumerate the events currently on the orphan list.

See also

NESLProduceEvent

NESLRegisterProducer
NESLDeRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLScanProducersByName
NESLScanConsumersByName

NESLProduceEvent

Description

An event producer calls this to notify registered consumers that the event has occurred.

Syntax

```
UINT32
NESLProduceEvent(
        NESL_ECB    *ProducerNecb,
        NESL_ECB    **ConsumerNecb,
        void        *eventData );
```

Parameters

ProducerNecb Pointer to NESL_ECB used during **NESLRegisterProducer**.

ConsumerNecb Points to a pointer to the NESL_ECB of the consumer who consumed the event. NULL if the producer does not care who consumed the event.

eventData Event-specific parameters. If more than a single data item needs to be passed, a pointer to an array is passed.. Otherwise, the single value is passed.

Return values

NESL_PRODUCER_NOT_FOUND
The producer is NULL.

NESL_EVENT_CONSUMED
Event is consumable and is consumed. *Consumer* set to the consumer's NESL_ECB.

NESL_EVENT_NOT_CONSUMED
Event is consumable and is not consumed. Consumer set to NULL.

NESL_EVENT_BROADCAST
Event has been broadcast to all consumers. Consumer not changed.

Remarks

If the event is consumable, then one of the consumers may consume the event, and event notification will stop.

If the producer and consumer routines are running on asynchronous events (e.g., IPX packets, interrupts), then the routines must be reentrant. **NESLProduceEvent** will not protect the consumer routine from being reentered.

For example, if the consumer routine reenables interrupts, another asynchronous event can be issued from the producer and thus reenter the consumer. It is up to either the producer or the consumer routine to protect themselves from reentrancy issues. Further, they must take steps to ensure that there is no stack overflow because of their activities.

See also

NESLRegisterProducer
NESLDeRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLEnumerateEvents
NESLScanProducersByName
NESLScanConsumersByName

NESLRegisterConsumer

Description

Registers the consumer of an event. If a producer of the event is not currently registered, the consumer is placed onto an orphaned consumer list.

Syntax

```
UINT32
NESLRegisterConsumer(
        NESL_ECB      *Consumer);
```

Parameters

Necb Points to NESLEventControlBlock.

NecbNext RESERVED. This field should not be modified by the calling routine while the NESL_ECB is registered.

NecbVersion

The support level expected by the application. This field allows the interface to be expanded while still providing full backward compatibility.

NecbOsiLayer

Determines the ordering of registered consumers of the same event. The format of this field is 0xLRRR, where L is the number (0-7) corresponding to the OSI Layer and RRR (0-4095) is the relative order with other modules also registered on that layer. The relative ordering is useful when certain events require specific consumer ordering.

The defintion NESL_HOOK_FIRST may also be used in element *NecbOsiLayer*. This definition causes a consumer to be hooked first, no matter what. If the caller sets the low byte of *NecbOsiLayer* to ths value, the consumer will be hooked first in the consumer list. Normally NESL events will put lower layer identifiers before the hooked lead element. If another call is made specifying this definition an error will be returned to the caller and the element will not be added to the list.

NecbEventName

ASCIIZ name string of the event or class of events. This name has a maximum length of NESL_MAX_NAME_LENGTH.

NecbRefData

RESERVED. The value of this field will be ignored.

PNecbNotifyProc

This field is a pointer to the event notification callback procedure.

```
UINT32
MyNotifyProc(
    NESL_ECB    *ConsumerNecb,
    NESL_ECB    *ProducerNecb,
    void        *eventData );
```

ConsumerNecb

Points to NESL_ECB used by consumer during **NESLRegisterConsumer**.

ProducerNecb

Points to NESL_ECB used by producer during **NESLRegisterProducer**.

eventData

If the producer only has one data item, it can be passed to the consumer as an argument or as an address.

If the producer has more data than one item or if the producer wishes to guarantee portability, then the address of an array of the data items should be passed. The structure of the *eventData* must be defined by the producer and known by the consumer if it is to be interpreted properly.

Return from a Consumer after an event notification callback:

NESL_EVENT_CONSUMED

Event was consumed by the consumer process

NESL_EVENT_NOT_CONSUMED

Event was not consumed by the consumer process

NOTE: This is only really applicable if the event is consumable, but a consumer should always do this to be compatible with both types of events. Called from foreground time or from interrupt time with interrupts enabled or disabled.

NecbOwner
Specifies the owner of the NESL_ECB. This field is platform-specific and platform-dependent. The DOS/MS Windows implementation *requires* this field to be set to the owner's module handle information.

NecbWorkSpace
RESERVED. This field should not be modified by the calling routine while the NESL_ECB is registered.

Return values

NESL_OK Registry succeeded.
NESL_EVENT_TABLE_FULL
The event was not registered because the event table is full.
NESL_DUPLICATE_NECB
The NESL Event Control Block was previously registered in the event table.
NESL_INVALID_NOTIFY_PROC
The consumer's notification procedure is NULL.
NESL_CONSUMER_NOT_FOUND
The NESL Event Control Block is NULL.
NESL_FIRST_ALREADY_HOOKED
The head of the consumer list has already been hooked.

Remarks

Called from foreground with interrupts enabled.

See also

NESLDeRegisterConsumer
NESLRegisterProducer
NESLDeRegisterProducer
NESLProduceEvent
NESLEnumerateEvents
NESLScanProducersByName
NESLScanConsumersByNamey

NESLRegisterProducer

Description

Registers the producer of an event and creates a consumer list containing the consumers of this event currently on the orphan list.

Syntax

```
UINT32
NESLRegisterProducer(
        NESL_ECB      *Producer);
```

Parameters

Necb Points to NESLEventControlBlock

NecbNext RESERVED. This field should not be modified by the calling routine while the NESL_ECB is registered.

NecbVersion The support level expected by the application. This field allows the interface to be expanded in the future while still providing full backward compatibility.

NecbOsiLayer RESERVED. The value of this field will be ignored.

NecbEventName

ASCIIZ name string of the event or class of events. This name has a maximum length of NESL_MAX_NAME_LENGTH.

NecbRefData This is a flag field used to specify whether the event is unique or consumable. It also indicates the sorting order for callouts to registered consumers.

Consumers which are already on the orphan list WILL be sorted when a new producer is registered. All consumers that are registered after a producer is registered will be correctly sorted.

NESL_SORT_CONSUMER_BOTTOM_UP
Use bottom-up relative ordering on the consumer's *NecbOsiLayer* field in maintaining an ordered list of consumers requiring notification.

NESL_CONSUME_EVENT
The event can be consumed by one of the registered consumers. By default, an event is broadcast to all registered consumers.

This flag will cause a chaining effect among the consumers which will start with the first registered comsumer and proceed to the next until one of the comsumers consumes the event or the end of the consumer list is reached.

NESL_UNIQUE_PRODUCER
The producer of the event must be unique. If there is another producer registered with the same event string, then this call will fail. By default, there can be multiple producers of the same event.

This flag is used to prohibit multiple producers provided that this is the first producer registered.

PNecbNotifyProc
RESERVED. The value of this field will be ignored.

NecbOwner
Specifies the owner of the NESL_ECB. This field is platform-specific and platform-dependent. The DOS/MS Windows implementation REQUIRES this field to be set to the owner's module handle information.

NecbWorkSpace
RESERVED. This field should not be modified by the calling routine while the NESL_ECB is registered.

Return values

NESL_OK Registry succeeded

NESL_REGISTED_UNIQUE
A previous producer has registered the event as unique and this producer tried to register the event as non-unique.
NESL_REGISTERED_NOT_UNIQUE
A previous producer has registered the event as non-unique and this producer tried to register the event as unique.
NESL_REGISTERED_CONSUMABLE
A previous producer has registered the event as consumable and this producer tried to register the event as broadcast.
NESL_REGISTERED_BROADCAST

A previous producer has registered the event as broadcast and this producer tried to register the event as consumable.

NESL_EVENT_TABLE_FULL
The event was not registered because the event table is full.

NESL_DUPLICATE_NECB
The NESL Event Control Block was previously registered in the event table.

NESL_PRODUCER_NOT_FOUND
The NESL Event Control Block is NULL.

Remarks

The event producer defines the rules necessary concerning process and interrupt time execution.

Called from foreground with interrupts enabled.

See also

NESLDeRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLProduceEvent
NESLEnumerateEvents
NESLScanProducersByName
NESLScanConsumersByName

NESLScanConsumersByName

Description

Returns a copy of the *Consumer* structure to the caller.

Syntax

```
UINT32
NESLScanConsumersByName(
        void            *ContextHandle,
        MEON_STRING     *EventName,
        NESL_ECB        *Consumer );
```

Parameters

ContextHandle Context of the previous call to **NESLEnumerateEvents**. Equals 0 if this is the first call.

EventName ASCIIZ event name.

Consumer Pointer to a valid NESL_ECB structure ready to receive a copy of the consumer's information.

Return values

NESL_OK The copy was successful.

NESL_CONSUMER_NOT_FOUND
The event is not found on the active list.

NESL_INVALID_DESTINATION
There is no destination.

NESL_INVALID_CONTEXT_HANDLE
The previous context handle could not be located.

NESL_CONSUMER_NOT_FOUND
There are no more consumers to be found.

Remarks

If *ContextHandle* is NULL, the first registered event is used. Otherwise, **ContextHandle* specifies the previous context and the next event is used.

Note: The caller *must* provide an allocated structure to copy into. This call will *not* enumerate the events currently on the orphan list.

Note: Scanning events must be done from a single thread or context of execution. Because any number of consumers or producers may register or deregister events when the foreground is relinquished, relinquishing control will invalidate the *ContextHandle*

See also

NESLProduceEvent
NESLRegisterProducer
NESLDeRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLEnumerateEvents
NESLScanProducersByName

NESLScanProducersByName

Description

Returns a copy of the *Producer* structure to the caller.

Syntax

```
UINT32
NESLScanProducersByName(
        void            *ContextHandle,
        MEON_STRING     *EventName,
        NESL_ECB        *Producer);
```

Parameters

ContextHandle Context of the previous call to **NESLEnumerateEvents**. Equals 0 if this is the first call.

EventName ASCIIZ event name.

Producer Pointer to a valid NESL_ECB structure ready to receive a copy of the producer's information.

Return values

NESL_OK The copy was successful.

NESL_PRODUCER_NOT_FOUND
The event was not found in the active list.

NESL_INVALID_DESTINATION
There is no destination.

NESL_INVALID_CONTEXT_HANDLE
The previous context handle could not be located.

NESL_PRODUCER_NOT_FOUND
There are no more producers to be found.

If *ContextHandle* is NULL, the first registered event is used. Otherwise, **ContextHandle* specifies the previous context and the next event is used.

The caller *must* provide an allocated structure to copy into.

This call will *not* enumerate the events currently on the orphan list.

Note: Scanning events must be done from a single thread or context of execution. Because any number of consumers or producers may register or deregister events when the foreground is relinquished, relinquishing control will invalidate the *ContextHandle*.

See also

NESLProduceEvent
NESLRegisterProducer
NESLDeRegisterProducer
NESLRegisterConsumer
NESLDeRegisterConsumer
NESLEnumerateEvents
NESLScanConsumersByName

NESL NecbRefData Flags

This list of flags are used by producers to define their NecbRefData.

```
#define NESL_NOT_UNIQUE_PRODUCER          0x00000000
#define NESL_BROADCAST_EVENT              0x00000000
#define NESL_SORT_CONSUMER_BOTTOM_UP      0x00000001
#define NESL_SORT_CONSUMER_TOP_DOWN       0x00000000
#define NESL_CONSUME_EVENT                0x00000002
#define NESL_UNIQUE_PRODUCER              0x00000004
```

NESL Return Codes

The return codes are all enumerated with the exception of NESL_EVENT_CONSUMED which is simply an alias for NESL_OK. See the function descriptions for a discussion of the meaning of any particular return code.

```
#define NESL_EVENT_CONSUMED   NESL_OK

enum {
    NESL_OK = 0,
    NESL_EVENT_NOT_CONSUMED,
    NESL_EVENT_BROADCAST,
    NESL_EVENT_NOT_REGISTERED,
    NESL_EVENT_TABLE_FULL,
    NESL_EVENT_IS_CONSUMABLE,
    NESL_EVENT_IS_NOT_CONSUMABLE,
    NESL_NO_MORE_EVENTS,
    NESL_PRODUCER_NOT_FOUND,
    NESL_CONSUMER_NOT_FOUND,
    NESL_INVALID_CONTEXT_HANDLE,
    NESL_INVALID_DESTINATION,
    NESL_REGISTERED_UNIQUE,
    NESL_REGISTERED_NOT_UNIQUE,
    NESL_REGISTERED_CONSUMABLE,
    NESL_REGISTERED_BROADCAST,
    NESL_REGISTERED_SORT_TOP_DOWN,
    NESL_REGISTERED_SORT_BOTTOM_UP,
    NESL_DUPLICATE_NECB,
    NESL_INVALID_NOTIFY_PROC,
    NESL_FIRST_ALREADY_HOOKED
);
```

PNecbNotifyPROC

A typedef which defines the callback function for consumer NESL_ECBs.

```
typedef UINT32 (*PNecbNotifyPROC)(
    NESL_ECB   *consumerNecb,
    NESL_ECB   *producerNecb,
    void       *eventData );
```

NESL OSI Layer Definitions

These definitions are used for the NESL_ECB structure element *NecbOsiLayer*.

```
#define NESL_APPLICATION_LAYER      0x7000
#define NESL_PRESENTATION_LAYER     0x6000
#define NESL_SESSION_LAYER          0x5000
#define NESL_TRANSPORT_LAYER        0x4000
#define NESL_NETWORK_LAYER          0x3000
#define NESL_DATALINK_LAYER         0x2000
#define NESL_PHYSICAL_LAYER         0x1000
#define NESL_NIOS_LAYER             0x0000
```

The following defintion is also used in element *NecbOsiLayer*:

The defintion NESL_HOOK_FIRST may also be used in element *NecbOsiLayer*. This definition causes a consumer to be hooked first, no matter what. If the caller sets the low byte of *NecbOsiLayer* to ths value, the consumer will be hooked first in the consumer list. Normally NESL events will put lower layer identifiers before the hooked lead element. If another call is made specifying this definition an error will be returned to the caller and the element will not be added to the list.

NESL_ECB Structure

```
typedef struct NECBStruct

struct
NECBStruct *NecbNext;
    UINT16          NecbVersion;
    UINT16          NecbOsiLayer;
    MEON_STRING     *NecbEventName;
    UINT32          NecbRefData;
    UINT32          (*PNecbNotifyProc)(
        struct      NECBStruct *consumerNecb,
        struct      NECBStruct *producerNecb,
        void        *eventData);
        void        *NecbOwner;
        void        *NecbWorkSpace;
    } NESL_ECB;
```

NecbNext A link pointer used by NESL to link NESL_ECBs together into processing lists.

NecbVersion Identifies the support level expected by the application. This field allows the interface to be expanded in the future while still providing full backward-compatibility.

NecbOsiLayer A value used in sorting linked lists of consumer NESL_ECBs.

This field is used to sort the consumers according to the producer's specifications (either top-down or bottom-up). The first nibble should be the OSI layer number (0 - 7). The remaining 3 nibbles should define the relative order with other consumers registered at the same OSI level. (i.e., 0xLRRR) This relative ordering is useful for several event consumers on the same OSI level which require a specific processing order for the event.

NecbEventName A pointer to an ASCIIZ string defining the name of the event or class of events. This name may have a maximum length of NESL_MAX_NAME_LENGTH.

NecbRefData Flags specified by producer NESL_ECBs defining whether the produced event is consumable or broadcast, and whether the producer is to be a unique producer or if other producers can register.

If the NESL_ECB is a consumer, this field is not used by NESL.

PNecbNotifyProc The callback routine registered by a consumer of an event. If the NESL_ECB is a producer, this field is not used by NESL.

NecbOwner A platform-specific field which identifies the owner module of the NESL_ECB. The DOS/Windows platforms use the module handle of the NLM to specify the owner.

NecbWorkSpace Field used by NESL to keep track of processing lists. This field must not be used by the owner module while the NESL_ECB is registered with NESL.

NESL_MAX_NAME_LENGTH

The maximum length (including the NULL terminator) of a *NecbEventName*. A buffer of this size must have been allocated to receive the return value from **NESLEnumerateEvents**.

```
#define  NESL_MAX_NAME_LENGTH        81
```