Chapter 7
NIOS Structures, Definitions, and Events

NIOS Structures and Variables

NiosCountryInfo

This structure is returned by **NiosGetCountryInfo**.

```
typedef struct NiosCountryInfoStruc

   UINT32 NCICountryCode;     // Rev 1 - As defined for DOS
   UINT32 NCICodePage;        // Rev 1 - As defined for DOS
   UINT32 NCIDateFormat;      // Rev 1 - See DATE_?? below
   UINT32 NCITimeFormat;      // Rev 1 - See TIME_?? below
   UINT8  NCIDateSeparator[2] // Rev 1 - ASCIIz string
   UINT8  NCITimeSeparator[2] // Rev 1 - ASCIIz string
} NiosCountryInfo;


      #define DATE_USA     0x00000000  // Month Day Year
      #define DATE_EUROPE  0x00000001  // Day Month Year
      #define DATE_JAPAN   0x00000002  // Year Month Day

      #define TIME_12HOUR  0x00000000  // 12 hour clock
      #define TIME_24HOUR  0x00000001  // 24 hour clock
```

DebInfoStruc

DebInfoStruc holds the debug state at the time the debug query is entered. A pointer to this structure is passed in the DebStatusStruc which is passed to the consumer handler.

Some debuggers, such as Nu-Mega's *Soft-Ice for Windows*, do not support this structure.

```
#include <niosdeb.h>
typedef struct DebInfoStruc

   UINT32   DISEAX;
   UINT32   DISEBX;
   UINT32   DISECX;
   UINT32   DISEDX;
   UINT32   DISESP;
   UINT32   DISEBP;
   UINT32   DISESI;
   UINT32   DISEDI;
   UINT16   DISES;
   UINT16   DISSS;
   UINT16   DISDS;
   UINT16   DISFS;
   UINT16   DISGS;
   UINT32   DISEIP;
   UINT16   DISCS;
   UINT32   DISEFlags;
   UINT32   DISCR0;
   UINT16   DISGDTLen;
   UINT32   DISGDTPtr;
   UINT16   DISPad0;
   UINT16   DISIDTLen;
   UINT32   DISIDTPtr;
   UINT16   DISPad1;
   UINT32   DISCR1;
   UINT32   DISCR2;
   UINT32   DISCR3;
   UINT32   DISDR0;
   UINT32   DISDR1;
   UINT32   DISDR2;
   UINT32   DISDR3;
   UINT32   DISUnknown0;
   UINT32   DISUnknown1;
   UINT32   DISDR7;
   UINT32   DISTR6;
   UINT32   DISTR7;
} DebInfo;
```

DebStatusStruc

DebStatusStruc holds the debug state at the time the debug query is entered. Two pointers are contained in this structure.

The first event-specific parameter is a pointer to the name of the module being queried. If a consumer matches this name it must process the query and then return, signalling that the event was consumed; else the consumer should signal that the query was *not* consumed.

The second event-specific parameter is a pointer to a **DebugInfoStruc** which holds the state of the processor at the time the debugger was invoked.

```
#include <debug.h>
typedef struct DebStatusStruc

   UINT8    *cmd;
   DebInfo  *DebugInfo;
} DebStatus;
```

FEBStruc

Control Block used by **NiosScheduleForegroundEvent**.

```
#include <nios.h>
typedef struct FEBStruc

   struct   FEBStruc *FEBLink;   //Modified by Nios
   UINT32   FEBReserved;         //Not modified
   UINT16   FEBStatus;           //!0=active, 0=active
   void  (*FEBESR)(struct FEBStruc *); //ESR
}FEB;
```

KeywordStruc

Structure defining a single keyword.

KeywordStr	Pointer to an uppercase ASCIIZ keyword.
KeywordLength	Number of bytes in keyword, including NULL byte.
KeywordFlags	Keyword's attributes.
KeywordActionProc	Pointer to routine that is called to process the keyword if it is encountered by the parser.
KeywordRefData	Value passed to **KeywordActionProc**() if the keyword is matched by the parser. It is also passed to **InvalidLineProc**() if there was a keyword match and a line overflow. This value can be anything the programmer desires.

LoadedModuleStruct

Structure holding information about a loaded NIOS Client module.

Offsets for most fields match the server's module structure.

```
#include <nios.h>
typedef struct LoadedModuleStruct

struct  LoadedModuleStruct *link
   UINT32   resourceCount;
   UINT32   totalAllocatedMemory;
   UINT32   reserved1;
   UINT32   languageID;                // -1 if no message file
   UINT8    *codeOffset;               // Paragraph aligned offset
   UINT32   codeSize;
   UINT8    *dataOffset;               // Paragraph aligned offset
   UINT32   dataSize;
   UINT32   uninitializedDataLength;
   UINT32   customDataOffset;
   UINT32   customDataSize;
   UINT32   loadAttributes;
   UINT32   moduleType;

   UINT32   (*initRoutine)( struct LoadedModuleStruct *moduleHandle,
            void    *screenHandle,
            UINT8   *commandLine,
            UINT8   *moduleLoadPath,
            UINT32  unitializedDataLength,
            UINT32  customDataFileHandle,

            UINT32  (*NiosRead)(
                     UINT32 customFileHandle,
                     UINT32 customOffset,
                     UINT8 *buf,
                     UINT32 UINT8sToRead),

            UINT32  customDataOffset,
            UINT32  customDataSize,
            UINT32  numMsgs,
            UINT8   **msgs);

   void     (*exitRoutine)( void);
   UINT32   (*checkRoutine)( void *screenID);
   UINT32   reserved2;
   UINT8    name[36];                   // Length preceded, filename
   UINT8    description[128];           // Length preceded, description
   UINT32   reserved3[5];
   UINT32   numReferencedModules;
   struct   LoadedModuleStruct **referencedModules;
                                   // -> array of modHandle's
   struct   VersionStampInfo version;
   UINT8    *copyright;             // Length preceded, copyright

}*modHandle, MOD_HANDLE, modHdlt, *ModHdlP;
```

VersionStampInfo

Version information stored in *LoadedModuleStruct.version.* Information found after the Client32 load header "VeRsIoN#" stamp.

```
#include <nios.h>
struct VersionStampInfo

   UINT32   majorVersion;
   UINT32   minorVersion;
   UINT32   revision;
   UINT32   year;
   UINT32   month;
   UINT32   day;
};
```

LoadedModuleStruct.loadAttributes Bit Definitions

```
#define MODULE_REENTRANT_FLAG       0x00000001
#define MODULE_MULTIPLE_LOAD_FLAG   0x00000002
#define MODULE_CANNOT_UNLOAD        0x20000000
#define MODULE_DEBUG_VERSION        0x40000000
#define MODULE_ALIGN_4K_PAGES       0x80000000 // NOT
                                                SUPPORTED
```

LoadedModuleStruct.moduleType Definitions

```
#define EXETYPE_NLM                 0x00000000
#define EXETYPE_LAN                 0x00000001
#define EXETYPE_PSEUDO              0xFFFFFFFF
```

MemInfoStruc

Structure returned by **NiosGetMemInfo**.

```
#include <nios.h>
typedef struct MemInfoStruc

    UINT32 MITotalSysFree;
    UINT32 MITotalSubFree;
    UINT32 MILargestSubFreeBlock;
    UINT32 MITotalAlloced;
    UINT32 MIAllocOverhead;
    UINT32 MIAvgAllocSize;
    UINT32 MITotalPhysAlloced;
    UINT32   MILargestSysFreeBlock;
}MemInfo;
```

MITotalSysFree	Total amount of free memory in the system. This value includes MITotalSubFree.
MITotalSubFree	Total amount of free memory available from memory sub-allocator.
MILargestSubFreeBlock	Largest free memory block available from memory sub-allocator. Requests to allocate larger blocks than this value cause the memory allocator to obtain more memory from the system free pool.
MITotalAlloced	Total amount of memory allocated by NLMs.
MIAllocOverhead	Overhead per allocation.
MIAvgAllocSize	Average allocation size.
MITotalPhysAlloced	Total amount of physically contiguous memory currently allocated.
MILargestSysFreeBlock	Largest free memory block available from the system memory manager. This may or may not be larger than MILargestSubFreeBlock, however it typically is.

NDateTime

Structure used by **NiosGetDateTime** and **NiosSetDateTime** functions.

```
#include <nios.h>
typedef struct NDateTimeStruc

UINT8 NDTHour;          // (0-23)
UINT8 NDTMinute;        // (0-59)
UINT8 NDTSecond;        // (0-59)
UINT8 NDTReserved;

UINT8 NDTDay;         // (1-31)
UINT8 NDTMonth;         // (1-12)
UINT16 NDTYear;       // (1980-2079)
}NDateTime;
```

NiosAESECB

```
#include <aes.inc>

NiosAESECB      struc
   AESLink         dd      ?
   AESReserved0    dd      ?  ; Reserved for NIOS use
   AESStatus       dw      ?  ; Reserved for NIOS use
   AESESR          dd      ?  ; Not modified my NIOS
   AESReserved1    dd      ?  ; Reserved for NIOS use
   AESReserved2    dd      ?  ; Reserved for NIOS use
NiosAESECB      ends
```

Nios Statistics Condensed Table Form

This is the format of tables returned from NiosStatGetTable:

Description	Size
NiosStatTableVer	UINT32
Description	UINT8 *
Long Name	UINT8 *
Reserved	UINT32 [3]
Options	UINT32
NumStats	UINT32
UseFlag 0	UINT32
Index 0	UINT32
StatPtr 0	void *
StatDescriptionPtr 0	UINT8 *
UseFlag 1	UINT32
Index 1	UINT32
StatPtr 1	void *
StatDescriptionPtr 1	UINT8 *
UseFlag 2	UINT32
Index 2	UINT32
StatPtr 2	void *
StatDescriptionPtr 2	UINT8 *
...	
UseFlag n	UINT32
Index n	UINT32
StatPtr n	void *
StatDescriptionPtr n	UINT8 *

Nios Statistics Entry

This is the format for each counter entry in a Nios statistics table

```
typedef struct _nios_stat_entry {
   UINT32   StatUseFlag;   // One of StatUseFlag values, see below
   UINT32   Index;         // Well known index of counter
   void     *Stat;         // Pointer to statistic
   UINT8    *StatString;   // ASCIIz description, should be language enabled
} NIOS_STAT_ENTRY;
```

Nios Statistics StatUseFlag

Values for NIOS_STAT_ENTRY *StatUseFlag* field. These match the ODI_STAT_ values. Users of NIOS_STAT_UNTYPED must define the stat as UINT32 length preceded, and so forth.

```
typedef struct _wamcoUntyped {
   UINT32 length;
   UINT8  Bob;
} WAMCO_UNTYPED;

WAMCO_UNTYPED WamcoStat = { 1, 42 };

#define NIOS_STAT_NOT_USED     0xFFFFFFFF
#define NIOS_STAT_UINT32       0x00000000
#define NIOS_STAT_UINT64       0x00000001
#define NIOS_STAT_ASCIIZ       0x00000002
#define NIOS_STAT_UNTYPED      0x00000003
```

Bit flag for NIOS_STAT_ENTRY StatUseFlag field.

#define NIOS_STAT_RESETTABLE   0x80000000

NIOS_STAT_RESETTABLE   This counter can be reset by another application (one other than the registering module).

Nios Statistics Table

The following is the format of the table passed to NiosStatRegister:

```
typedef struct _nios_stat_table {
   UINT32   NiosStatTableVer; // Managed by the calling NLM
   UINT8       *Description;  // 'Well known' name of table
   UINT8       *LongName;     // long description of table
   UINT32   Reserved [ 3 ];   // Reserved for use by Nios
   UINT32   Options;          // see NIOS_STAT_TABLE options
   UINT32   NumStats;         // Number of stat entries
   NIOS_STAT_ENTRY   *Stats;  // Pointer to array of stat entries
} NIOS_STAT_TABLE;
```

Note: *LongName* should be language enabled. *Description* should not.

stdOutInfo

Structure used for **NiosRegisterStdOutHandler** and **NiosDeRegisterStdOutHandler** functions.

```
#include <nstdlib.h>
typedef struct stdOutStruc

struct stdOutStruc *SOILink;
UINT32 (*SOIHandler)(
      struct      stdOutStruc *stdOutBlock,
      modHandle   module,
      UINT8       *prefix,
      UINT8       *msg);
   void *SOINiosWorkspace;
}stdOutInfo;
```

TraceOut

```
#include <nios.inc>

TraceOut "String", parm1, parm2, ...
```

TraceOut is an assembly macro used to generate a trace message to the debug terminal if DEBUG is defined. No code is generated if DEBUG is *not* defined. This function preserves all registers.

This macro includes a *newline* at the end of *String*.

Note that parameters containing white space must be enclosed in brackets <>; for example, <offset mylabel>.

An NLM which uses this macro must include **NiosDprintf** in the modules's linker function import list.

VersionStampInfo

Version information stored in *LoadedModuleStruct.version*. Information found after the Client32 load header "VeRsIoN#" stamp.

```
#include <nios.h>
struct VersionStampInfo

   UINT32   majorVersion;
   UINT32   minorVersion;
   UINT32   revision;
   UINT32   year;
   UINT32   month;
   UINT32   day;
};
```

Bit Definitions, Return Codes, and Constants

AES Return Codes

```
#include <aes.inc>

AES_SUCCESS          equ   0
AES_ITEM_NOT_PRESENT equ   -1
```

msgType Values

```
#include <nstdlib.h>

#define MT_NOMSG                0x00000000
#define MT_INFORM               0x00000001
#define MT_INIT_FATAL           0x00000002
#define MT_ALERT                0x00000003
#define MT_FORCED_ALERT         0x00000004
#define MT_ABEND                0x00000006
#define MT_DEBUG_OUT            0x00000008
#define MT_DEBUG_TRACE          0x00000009
#define MT_NW_BROADCAST         0x0000000A
#define MR_LOG_STATUS           0x0000000B
```

All message are language enabled except where noted.

MT_NOMSG	Effectively an NOP. **NiosPrintf** ignores this message.
MT_INFORM	Used to display normal status information during an initialization routine. This message type cannot be used at interrupt time.
MT_INIT_FATAL	Used to display messages describing why a module was unable to initialize during a module's init routine. This message type cannot be used at interrupt time.
MT_ALERT	Used to display messages describing events that are abnormal, affecting the user in some way. These messages are queued and displayed at a later time. The user must acknowledge the message before continuing. This message type can be used at interrupt time.

MT_FORCED_ALERT	Used to display messages describing events that are abnormal, affecting the user in some way. Forced alerts are serviced immediatelly regardless of any critical sections or other reasons alerts are normally delayed. The user must acknowledge the message before continuing. This message cannot be used at interrupt time.
MT_ABEND	Immediately displays the message and hangs the system. This should be used when an unrecoverable event has occurred and system operation cannot continue reliably. Typically this is for the "never should happen" cases. This message type can be used at interrupt time.
MT_DEBUG_OUT	Message is displayed in the active debugger environment is equivalent to calling **NiosDprintf** function. This message type can be used at interrupt time. Debug messages are not required to be enabled. If a debugger is not present, it is interrupt time, and logging is enabled (see MT_LOG_STATUS), the message will be logged to the logfile. Otherwise, if a debugger is not present, this message will not be seen.
MT_DEBUG_TRACE	Message is placed in NIOS's trace buffer (if active). This message type can be used at interrupt time. Debug messages are not required to be enabled. If a debugger is not present, it is interrupt time, and logging is enabled (see MT_LOG_STATUS), the message will be logged to the logfile.

Note: NOT SUPPORTED YET.

MT_NW_BROADCAST	NetWare broadcast information message.

MT_LOG_STATUS

Message is timestamped and logged to Nios logfile. Logging can be turned on or off with the NiosEnableLogging api. The initial logging mode is OFF unless nios is loaded with the /L command line parameter. This message cannot be used at interrupt time.

Optional MT Flags, ORed with msgType Value

```
#define     MTF_PREFIX            0x80000000
#define     MTF_INDIRECT_ARGS     0x40000000
#define     MTF_NO_TIMESTAMP      0x20000000
```

MTF_PREFIX Specifies that the message should be prefixed with a prefix of the format "a-b-c:", where "a" is the module's name, "b" is the module's version, and "c" is the message ID of the base format string, if it exists in the module's message file. Typically this flag is used to display warning or fatal error messages.

MTF_INDIRECT_ARGS

Specifies that the first parameter after the formatStr parameter is a pointer to a list of arguments to use when processing the formatStr. If not specified, the list of arguments simply follow the formatStr parameter.

MTF_NO_TIMESTAMP

Supresses timestamp on logged messages.

NiosCharTable

NiosCharTable is an exported public table that contains type information about all 256 characters. Simply index into the table using the character value and mask with one of the type bits shown below.

```
#include <nios.h>

#define IS_NOT    0x00  //Nothing
#define IS_SPC    0x01  //White Space
#define IS_DIG    0x02  //Numerical digit 0-9
#define IS_HEX    0x04  //Hex digit 0-9, A-F, & a-f
#define IS_UPP    0x08  //Uppercase alpha letter
#define IS_LOW    0x10  //Lowercase alpha letter
#define IS_CTL    0x20  //Control character
#define IS_PUN    0x40  //Punctuation
#define IS_LDB    0x80  //Lead byte of a DBC char
```

NiosEnableLogging

Values for *NiosEnableLogging*:

```
#include <nstdlib.h>

#define  NIOS_LOG_DISABLE         0x00000000
#define  NIOS_LOG_ENABLE          0x00000001
```

NiosHookHardwareInt Option Value

Possible option value for **NiosHookHardwareInt** is as follows:

```
#define HIOPT_SHAREABLE_BIT       0x00000001
#define HIOPT_C_HANDER_BIT        0x00000002
```

Nios Statisitcs Get Options

Bit flag values for get options in the **NiosStatGetTable** API. Unused bits must be 0

```
#define NIOS_STAT_GET_OPTION_REFRESH   0x00000001
```

NIOS_STAT_GET_OPTION_REFRESH	Only update counters

Nios Statistics Max Name Length

Maximum length (including null) for a Nios Statistics stat or table description. This number was chosen arbitrarily to be <width of typical screen> - 10.

```
#define NIOS_STAT_MAX_NAME    70
```

Nios Statistics Status Codes

```
enum {
   NIOS_STAT_SUCCESS_CODE,
   NIOS_STAT_OUT_OF_CLIENT_MEMORY,
   NIOS_STAT_INVALID_PARAMETER,
   NIOS_STAT_NOT_REGISTERED,
   NIOS_STAT_BUFFER_TOO_SMALL,
   NIOS_STAT_NO_MORE_TABLES,
   NIOS_STAT_READ_ONLY
};
```

Nios Statistics Table Options

Bit flag values for NIOS_STAT_TABLE Options field. Unused bits must be 0.

```
#define NIOS_STAT_TABLE_HAS_RESETTABLE  0x00000001
```

NIOS_STAT_TABLE_HAS_RESETTABLE
Table contains >= 1 resettable counter

Nios Type Values

The following values are returned by **NiosGetVersion**:

```
#include <nios.h>

#define NIOS_FOR_DOSWIN_VMM   0
#define NIOS_FOR_WIN4X_VMM    1
#define NIOS_FOR_NETWARE_OS   2
```

Popup Video Definitions

```
/* Message Box support ----------------------------*/
         #define MB_OK                     0x0000
         #define MB_OKCANCEL               0x0001
         #define MB_ABORTRETRYIGNORE       0x0002
         #define MB_YESNOCANCEL            0x0003
         #define MB_YESNO                  0x0004
         #define MB_RETRYCANCEL            0x0005
         #define MB_SYSTEMMODAL            0x1000


/* Standard dialog button IDs -------------------*/
         #define IDOK                      1
         #define IDCANCEL                  2
         #define IDABORT                   3
         #define IDRETRY                   4
         #define IDIGNORE                  5
         #define IDYES                     6
         #define IDNO                      7
```

NIOS_MAX_PROCESS_GROUPS

Defines the maximum number of possible execution environments. An execution environment is made up of one or more processes that share the same set of resources (for example network drives).

See also: NiosGetCurrProcessGroupId

```
#include <nios.h>

#define NIOS_MAX_PROCESS_GROUPS      32
#define NIOS_SYS_PROCESS_GROUP_ID    1
```

MAX_PROCESS_NAME_LEN

Required length of the retBuf array passed to the **NiosGetProcessName** service.

```
#define  MAX_PROCESS_NAME_LEN    256  // Includes NULL
```

Nios Events

NIOS Module Preload/Preunload

```
#include <nios.h>

#define  NE_MODULE_PRELOAD    "NIOS MODULE PRELOAD"
#define  NE_MODULE_PREUNLOAD  "NIOS MODULE PREUNLOAD"
```

These events are consumable. If consumed, the operation is aborted. A pointer to the module handle of the module being loaded or unloaded is passed to the consumer as the first event-specific parameter (that is, the first parameter after the *ProducerNecb* parameter.

NIOS Module Loaded/Unloaded

```
#define  NE_MODULE_LOADED     "NIOS MODULE LOADED"
#define  NE_MODULE_UNLOADED   "NIOS MODULE UNLOADED"
```

These events are not consumable. A pointer to the module handle of the module being preloaded or preunloaded is passed to the consumer as the first event-specific parameter.

NIOS Idle

```
#define  NE_IDLE     "NIOS IDLE"
```

This event is generated when a state of idleness has been detected in the system. This is a consumable event. Consumers can use this event to perform background work.

Consumers should consume the event if they perform significant processing during the event.

Interrupt state is undefined when this event is generated. If a consumer of this event is going to perform some action, it should make sure interrupts are enabled.

The frequency of this callout is environment-dependent; consumers should therefore schedule an AES event as a fallback mechanism to perform their background work in case this event does not occur frequently enough.

NIOS Debug Query

```
#include <debug.h>

#define  NE_DEBUG_QUERY    "NIOS DEBUG QUERY"
```

NE_DEBUG_QUERY is generated when the user queries a system component's internal debug information. Each NLM that wishes to provide a debug query facility should register to receive this event.

A single event-specific parameter is passed to the event handler when the user queries a component. This parameter is a pointer to a structure that in turn contains two pointers to debug-specific data structures as described below.

The first event-specific parameter is a pointer to the name of the module being queried. If a consumer matches this name it must process the query and then return signaling that the event was consumed, or else the consumer should signal that the query was not consumed.

The second event-specific parameter is a pointer to a **DebugInfoStruc** that holds the state of the processor at the time the debugger was invoked.

Consumers must process this event with interrupts disabled.

Nios Stat Reset Table

Produced by NIOS when a statistics table is reset. A pointer to the null termianted well known name of the table is passed as the first event specific parameter. This name must be treated as read only. Note that not all statistics can be reset, depending on the use of the NIOS_STAT_RESETTABLE and NIOS_STAT_TABLE_HAS_RESETTABLE flags.

```
#define  NE_STAT_RESET_TABLE   "NIOS_STAT_RESET_TABLE"
```

See Also: NiosStatRegister, NiosStatDeRegister, NiosStatEnumerate, NiosStatGetTable, NiosStatResetTable

Nios Pre Reboot Notify

This event is generated when the user presses Ctrl-Alt-Del. This is not a consumable event. Consumers can use this event to perform operations needed prior to system reset. Interrupts are enabled.

```
#define  NE_PRE_REBOOT_NOTIFY "NIOS PRE REBOOT NOTIFY"
```

NIOS Process Group Create

This event is generated when a group of one or more processes are created that share resources. This is *not* a consumable event. The first custom parameter contains the process group ID assigned to this process group.

#define NE_PROCESS_GROUP_CREATE   "NIOS PROCESS GROUP CREATE"

NIOS Process Group Destroy

This event is generated when a group of one or more processes that share resources is destroyed. This is *not* a consumable event. The first custom parameter contains the process group ID of the process group which is being destroyed. Note that after an ID is destroyed, it may be reused for a new process group.

#define NE_PROCESS_GROUP_DESTROY "NIOS PROCESS GROUP DESTROY"

NIOS Process Create

This event is generated when a process in the system is being created. This is NOT a consumable event. The event data parameter passed to the consumer points to a ProcInfoStruc as shown below. Note that this event is generated for both Ring 3 user level processes as well as NLM modules.

When a NLM loads, this event is generated with the PDProcessGroupId field set to PROCESS_GROUP_NLM and the PDProcessId field set to the NLM's module handle.

Note: This event is not generated when a DOS real mode program is executed unless the 32-bit NetWare Shell is installed.

#define NE_PROCESS_CREATE   "NIOS PROCESS CREATE"

ProcInfoStruc

```
typedef struct
  UINT32   PDProcessGroupId;
  UINT32   PDProcessId;
}ProcInfoStruc;
```

NIOS Process Destroy

This event is generated when a process in the system is being destroyed. This is NOT a consumable event. The event data parameter passed to the consumer points to a ProcInfoStruc as shown above. Note that this event is generated for both Ring 3 user level processes as well as NLM modules.

When a NLM unloads, this event is generated with the PDProcessGroupId field set to PROCESS_GROUP_NLM and the PDProcessId field set to the NLM's module handle.

```
#define  NE_PROCESS_DESTROY        "NIOS PROCESS DESTROY"
```

Process Group Id for NLMs

This is a pseudo process group Id. The NIOS events "NIOS PROCESS GROUP DESTROY" and "NIOS PROCESS GROUP CREATE" are NOT generated for this pseudo Id.

```
#define  PROCESS_GROUP_NLM    0xFFFFFFFE
```

NIOS Reboot Notify

This event is generated when the user presses Ctrl-Alt-Del. This is *not* a consumable event. Consumers can use this event to perform operations needed prior to system reset. Interrupts are disabled.

```
#define NE_REBOOT_NOTIFY       "NIOS REBOOT NOTIFY"
```