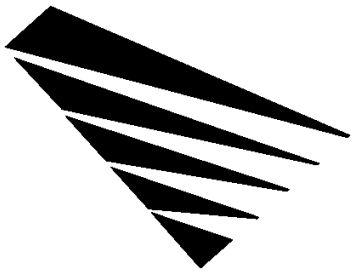

Chapter 9
Network Configuration File

NETCFG Overview

The NIOS Client expects a configuration file named NET.CFG to be located in the directory where NIOS was loaded. Because the format of the configuration file is specific to the environment, the only way for an NLM to be OS-independent is to use the NiosCfg API functions.

The format of the system configuration file is shown in this appendix.

The format is outlined as follows:

```
<section name>
    <configuration keyword name> [[=] <value>]
        this is comment
    <configuration keyword name> [[=] <value>] ; comment
    <configuration keyword name> [[=] <value>]
    <configuration keyword name> [[=] <value>]
    <configuration keyword name> [[=] <value>]
this is a comment at the beginning of a line
```

A section configuration block is a group of statements that describes the current configuration of a driver. These configuration statements are line-oriented: a statement cannot be continued on another line.

Declaration Statements

The first line of a section configuration block is the section configuration declaration statement. The declaration simply states the name of the driver to be configured.

The declaration statement syntax is <section name> and must start at column 1 of the line. Any section declaration which does not start at column 1 will not be recognized. Examples of section declaration statements are:

```
IPX         ; declaration of driver IPX

NE2000  ; declaration of driver NE2000

LSL         ; declaration of driver LSL
```

Keyword Statements

The body of the section configuration block is made up of a list of configuration keyword statements. Keyword statements must be indented at least one white space (space, tab) right of the driver declaration statement. Any keyword statements starting at column 1 of a line will interpreted (incorrectly) as beginning a new driver configuration block.

The syntax for keyword statements is

```
<configuration keyword name> [[=] <value>].
```

Keyword statements consist of a keyword name followed by an optional assignment operator ('='), followed by zero or more optional parameter values. Keyword names are defined by the specific module.

The optional assignment operator ('=') can be used to indicate assignment, but need not be present: It is equivalent to a white space.

Parameter Values

The parameter assigned to a keyword may be anything (such as a string, number, or boolean value). *Internally* the parameter is a string. The string can be (optionally) converted to a number of different formats, depending on the NLM, or the NLM can convert the value itself.

Parameter values that are string literals must have double quotes around them. String literals are useful for case-sensitive values or values that have an embedded '=' or ';' character. Embedding a double quote character within a string literal is not allowed.

Whenever a parameter value string is read by the parser functions, the value string is converted to uppercase. The only exception to this is when lowercase characters are contained within string literals. Uppercasing is done to save the driver from the effort of uppercasing all strings before interpretation.

String literal values passed to a driver will have the string delimiters ("") included to help the driver distinguish a string literal from normal string value.

Some keyword statements need no parameter value; the mere presence of the keyword name indicates a feature which should be enabled/disabled in the driver.

Example driver configuration block

```
x                           ;declaration - driver name
XParam 1 = TRUE             ;parameter - assign with option '='
Xparam 2 23                 ;parameter - assign without '='
XParam 3 "Wizard of OZ"   ;parameter - string literal
XParam 4 "CASE STRING"      ;parameter - case sensitive string literal
XParam 5 "case STRING"      ;parameter - case sensitive string literal
XParam 6 no case string   ;parameter - case insensitive string
XParam 7 A B "C D" E      ;parameter - mixed strings
XParam 8                    ;parameter - no value
NetAddr FADE2300          ;parameter - example of network address
NodeAddr 00001B31AD13       ;parameter - example of node address
```

In this example, parameters can have multi-word names and multi-word values. The line "XParam 6 no case string", for example, has a multi-word name "XParam 6" followed by a multi-word value "no case string". These features provide a high degree of flexibility when defining a configuration block for a driver.

Comments

Comments can be place virtually anywhere in the file. Comments are line-oriented and are delimited by the semicolon ';' character. Once a semicolon is encountered, the rest of the line is ignored.

Comments that start at column 1 inside a driver configuration block will not cause block termination. For example:

```
    Y1 TRUE
This comment does not cause block termination
    Y2 FALSE
```

Parsing the File

When parsing the configuration file, the parser will read one line at a time. The parser reads a *physical line* into a *logical line* buffer. A logical line is a line which has been stripped of comments, preceding white spaces, back-end white spaces, '=' characters, and extra white spaces.

The format of a logical line is as follows:

```
    <ParamName><SPACE><ParamValueStr>
```

The maximum length of a logical line is defined by NC_MAX_LINE_LEN. The maximum length of the parser's logical line buffer is NC_MAX_BUF_LEN.

NC_MAX_LINE_LEN and NC_MAX_BUF_LEN are independent of how many white spaces physically precede <ParamName> and <ParamValueStr> in the configuration file.

For instance, the following string

```
Parameter 1    =   This is an example
```

would logically translate to

```
Parameter 1 This is an example
```

The logical line length is

strlen("Parameter 1") + 1 + strlen("This is an example")

File IO APIs

Four APIs are used to read and write to a configuration file:

NiosCfgRead NiosCfgReadSpecific
NiosCfgWrite NiosCfgWriteSpecific

NiosCfgRead and **NiosCfgWrite** are simple APIs which operate on the system configuration file. They deal only with one keyword and driver name at a time, and will return the *first* match found for the driver name and keyword combination.

NiosCfgReadSpecific and **NiosCfgWriteSpecific** query the configuration file for section names and keywords that are not the first occurrence of each. These routines take an index as a parameter, and allow a wildcard match character (*).

Incrementing the index from zero allows **NiosCfgReadSpecific** to return keyword strings from multiple instances of the same keyword. If the wildcard match character is used in conjunction with an incrementing index, the whole net configuration database may be enumerated.

Configuration Definitions

Return Codes

NiosCfgRead

```
#define NC_OK                0x00000000 Operation succeeded
#define NC_LINE_OVERFLOW     0x00000001 Line overflow
#define NC_PARAM_NOT_FOUND   0x00000002 Keyword was not found
#define NC_TRUNCATED         0x00000004 Line was truncated to 255 characters
#define NC_OPEN_FAILED       0x00000008 Open of the cfg file failed
#define NC_ALLOC_FAILED      0x00000010 Allocation memory buffer(s) failed
#define NC_READ_FAILED       0x00000020 Reading the cfg file failed
```

NiosCfgWrite

```
#define NC_OK                0x00000000 Operation succeeded
#define NC_LINE_OVERFLOW     0x00000001 Line overflow
#define NC_PARAM_NOT_FOUND   0x00000002 Keyword was not found
#define NC_TRUNCATED         0x00000004 Line was truncated to 255 characters
#define NC_OPEN_FAILED       0x00000008 Open of the cfg file failed
#define NC_ALLOC_FAILED      0x00000010 Allocation memory buffer(s) failed
#define NC_READ_FAILED       0x00000020 Reading the cfg file

#define NC_WRITE_FAILED      0x00000040 Writing the new cfg file failed
#define NC_DELETE_FAILED     0x00000100 Delete of old cfg file failed
#define NC_RENAME_FAILED     0x00000200 Rename of new cfg file failed
#define NC_CREATE_FAILED     0x00000400 Create of new cfg file failed
```

ConvFlags Parameter

Possible values for **NiosCfgRead** and **NiosCfgWrite** ConvFlags parameter. These values are also used for **NiosKeywordRegister** and **NiosKeywordEnumerate**.

Only one of the following values can be specified.

```
#define  CFG_CONV_NONE          0x00000000
#define  CFG_CONV_STRING        0x00000001
#define  CFG_CONV_DEC_UINT32    0x00000002
#define  CFG_CONV_HEX_UINT32    0x00000003
#define  CFG_CONV_BOOLEAN       0x00000004
#define  CFG_CONV_DELETE        0x00000005
```

Option that can be ORed into **NiosCfgRead** Flags parameter.

```
#define  CFG_FLAG_ANYWHERE  0x80000000
```

Maximum Read/Write Line Lengths

```
#define NC_MAX_LINE_LEN     255     // Length of a keyword name string +
                                    one space + parameter value string

#define NC_MAX_BUF_LEN      NC_MAX_LINE_LEN+1

                                // Length of a keyword name string + one
                                space + parameter value string + null
                                terminator
```

Keyword Registry Return Codes

Defined return codes for modules:

NiosKeywordRegister
NiosKeywordDeRegister
NiosKeywordEnumerate
NiosKeywordSetValue
NiosKeywordResetValue
NiosKeywordUpdateNetCfg

```
#define   NC_INVALID_MODULE_HANDLE      0x1
#define   NC_OUT_OF_CLIENT_MEMORY       0x2
#define   NC_INVALID_CFG_MEMORY         0x3
#define   NC_NO_MORE_ENTRIES            0x4
#define   NC_KEYWORD_NOT_FOUND          0x5
#define   NC_KEYWORD_READ_ONLY          0x6
#define   NC_CLIENT_NOT_FOUND           0x7
#define   NC_KEYWORD_ALREADY_REGISTERED 0x8
#define   NC_GENERAL_ERROR              0x9
#define   NC_KEYWORD_INCOMPATIBLE       0xa
```

Keyword Registry Attribute Types

Defined attribute types for keyword registry:

```
#define   KEYWORD_READ_WRITE          0x0
#define   KEYWORD_READ_ONLY           0x1
#define   MAX_KEYWORD_ATTRIBUTE       KEYWORD_READ_ONLY
```

NIOS Configurable Parameters

Certain NIOS parameters are configurable in the NET.CFG file. This section describes the NIOS configurable parameters, and gives an example of the NET.CFG format.

All NIOS parameters are listed in NET.CFG under the NIOS section header. (See Appendix G for a complete description of the NET.CFG file.)

For example:

```
NIOS
    GLOBAL V86 WIN PAGES 5
    MIN MEM FREE AT WIN START 12800
    USE VIDEO BIOS ON
    PHYS MEM BELOW 16 MEG OFF
```

ALERT BEEP [ON|OFF]

Configuration parameter which configures NIOS.EXE to sound an audible beep when displaying popup alert messages.

The default is ON.

LINE DRAW CHARS "characters"

Configuration parameter which configures NIOS.EXE to use the specified line draw characters when displaying character mode popup messages. The characters must be in quotes and in the specified order.

character #1 TOP_LEFT_CORNER
character #2 TOP_RIGHT_CORNER
character #3 BOTTOM_LEFT_CORNER
character #4 BOTTOM_RIGHT_CORNER
character #5 VERTICAL_LINE
character #6 HORIZONTAL_LINE

MIN MEM FREE AT WIN START

Configuration parameter which specifies the minimum amount of NIOS free memory available when enhanced-mode MS Windows is loaded.

If the amount of free memory is less than this value, NIOS pre-allocates more memory from the XMS memory pool before continuing with Windows initialization.

This parameter is used to guarantee that a certain amount of memory is available for use by NLMs across an MS Windows session. For example, if an NLM that exhausts the NIOS memory pool is loaded inside of MS Windows, then NIOS must allocate more memory from Windows to allow the NLM to load.
The memory allocated from Windows cannot be used or accessed after Windows is exited; therefore users may not exit Windows unless they first unload the NLM or NLMs that are using Windows memory.

By specifying a large enough value for this parameter, NLMs can be loaded in a Windows session and still allow the user to exit back to DOS.

This parameter is automatically adjusted by NIOS when the user attempts to exit Windows and the exit is denied because MS Windows memory was allocated for NLM use during the Windows session. The value is adjusted upward so that the next time the user attempts the same configuration, exiting Windows will be possible.

PHYS CONTIGUOUS MEM [ON|OFF]

Determines whether memory allocated to LAN adapter drivers must be physically contiguous. This parameter must be set to ON when using a LAN adapter which uses any form of direct memory access (DMA). Memory can be used more efficiently if this parameter is set to OFF.

If this parameter is set to OFF, the "PHYS MEM BELOW 16 MEG" option is forced to OFF as well.

The default is ON.

PHYS MEM BELOW 16 MEG [ON|OFF]

Determines whether memory allocated to LAN adapter drivers must be below the 16 megabyte address boundary. This parameter must be set

to ON for using a LAN adapter which is incapable of accessing memory above the 16 megabyte boundary.

Adapters that have this constraint are 24-bit (ISA) cards which utilize direct memory access (DMA). Memory can be used more efficiently if this parameter is set to OFF.

The default is ON.

USE VIDEO BIOS [ON|OFF]

Configures NIOS.EXE to use BIOS or direct video memory access when displaying popups (for example, Alerts). If set to ON, NIOS uses BIOS calls; otherwise NIOS uses direct video memory access.

Direct video memory access is faster and is the default. If NIOS popups do not behave correctly on a system, setting this parameter to ON may eliminate the problem.