Appendix C LDI/DDI Specification Supplements

This Appendix contains excerpts from the specification supplements for the development of installation files for both LAN drivers (LDI) and Device Drivers (DDI).

What is the Installation Information File?

In order for a software utility to install MLIDs and disk drivers, it must know the parameters associated with each driver, the input required from the user, and how to set up the configuration file(s).

The installation information file for an MLID or a disk driver describes the configurable driver parameters, the input required from the user, and the format of the required output.

The information file contains one or more driver descriptions. Each driver description refers to a primary driver file and can also refer to other auxiliary driver files. Multiple descriptions in multiple files can also refer to a single driver. During installation, the installation information and the referenced driver file(s) are copied to a permanent directory on the machine's hard drive.

The NetWare Server Installation Information File

The driver installation information is contained in ASCII text files that are shipped with the driver. The installation information filename must have one of the following extensions, depending on the type of driver it references:

Installation Information Filename Extensions

Environment	Extension
Server LAN Driver	.LDI
Server Disk Driver	.DDI

Reserved Keywords

The keywords listed in the following table have special meaning in the installation information file. This supplement describes, in the appropriate section, each of the keywords listed in the table.

Note: This supplement uses the full form of the keywords for clarity; however, you should use the abbreviated form when you create the final installation information file. You should also use whitespace and comments sparingly.

Installation Information File Keywords and Abbreviations	
Keyword	**Abbreviation**
AND	**AND**
CDESCRIPTION	**CD**
CHOICE	**CH**
CPROG	**CP**
DECIMAL	**DEC**
DEFAULT	**DEF**
DESCRIPTION	**DES**
DLANGUAGE	**DLANG**
DRIVER	**DR**
ELSE	**ELS**
FILE	**FILE**
FRAME	**FR**
HELP	**HELP**
HEX	**HEX**
HIDDEN	**HID**
IF	**IF**
LANGUAGE	**LANG**
LIST	**LI**
NOT	**NOT**
OCTETBITORDER	**OCT**
OFILES	**OF**
OPTIONAL	**OPT**
OR	**OR**

OUTPUTFORMAT	**OUT**
PARAMETERVERSION	**PAR**
PATH	**PATH**
PRODUCTID	**PROD**
PROMPT	**PR**
REQUIRED	**REQ**
RESERVEDLENGTH	**RES**
STRING	**STR**
SYNTAXVERSION	**SYN**
TIMEOUT	**TIME**
TYPE	**TYP**
UNDEFINED	**UND**
VALUES	**VAL**
VERSION	**VER**

Information File Format and Syntax

Format of the Installation File

A driver information file contains one or more driver descriptions, as well as language definitions for the strings within the descriptions. The general format of the description file is shown below:

```
;DrIvEr DeScRiPtIoN

  Version: <driver description file number>
  SyntaxVersion: 1.00

  Driver <driver description #1 name> <dependency expression>
  {
    <driver #1 description, may include $<string> variables>
  }

  Driver <driver description #2 name> <dependency expression>
  {
    <driver #2 description, may include $<string> variables>
  }
  .
  .
```

```
   .

DLanguage: <default language ID>
   $<string #1 variable name> = "<string #1 text in default
language>"
   $<string #2 variable name> = "<string #2 text in default
language>"
   .
   .
   .

Language: <language #1 ID>
   $<string #1 variable name> = "<string #1 text in language #1>"
   $<string #2 variable name> = "<string #2 text in language #1>"
   .
   .
   .
Language: <language #2 ID>
   $<string #1 variable name> = "<string #1 text in language #2>"
   $<string #2 variable name> = "<string #2 text in language #2>"
   .
   .
   .
;DrIvEr DeScRiPtIoN EnD
```

The following section describes each portion of the above example.

The Initial and Final Lines

Note the initial and final lines:

```
;DrIvEr DeScRiPtIoN
.
.
.
;DrIvEr DeScRiPtIoN EnD
```

These signature lines bracket the driver installation information. The installation/configuration utility searches for these signatures whenever an information file is appended to the driver module. These lines are required and *must appear exactly as shown*. They must not be translated to another language.

Version

The *Version* label is optional, and the installation/configuration utility ignores it. The *Version* label allows you to manually control the version number.

SyntaxVersion

The *SyntaxVersion* label is mandatory. *SyntaxVersion* informs the installation/configuration parser of the syntax to expect in the driver information file. Novell controls the syntax version number; the version is currently 1.00.

Driver Section

The driver section includes one or more driver information blocks. Each driver block contains a short description of the driver, help information, and the driver's configurable parameters. For an explanation of the syntax within the driver block see the ``Driver Section."

Language Section

The language section allows you to translate text strings (help messages, prompts, etc.) to different languages. Language translation is optional and is not necessary for drivers that will operate only in a single language.

If you implement the language section, the driver references any translatable text strings with string names. Each language block then contains the string names and the corresponding text in the respective language.

An example driver information file is shown below.

Example

```
;DrIvEr DeScRiPtIoN
Version: 1.00
SyntaxVersion: 1.00

;File SAMPLE1.INF
  Driver SAMPLE1
 {
      Description:          $DESCRIPTION
   Help:              $HELP

  PROMPT INT
 {
      Values:               3, 5, 7, 9
   Default:          3
  }

  PROMPT PORT
  {
         Values:            300, 310, 320
```

```
        Default:        300
    }

    FRAME FrameSelect
    {
           Help:           "The driver defaults to using

                     the 802.3 frame type. You can
                remove this frame type and/or add
        the 802.2, 802.2 SNAP, or
     Ethernet II frame types."

        CDescription:     "802.3"
        Choice:         "Ethernet_802.3"

        CDescription:     "802.2"
        Choice:         "Ethernet_802.2"

        CDescription:     "802.2 SNAP"
        Choice:         "Ethernet_SNAP'

        CDescription:     "Ethernet II"
        Choice:         "Ethernet_II"

        Default:         1
    }
}
DLanguage: 4
; Default English
  $DESCRIPTION =      "Sample driver description"
  $HELP =             "Sample help text information"

Language: 247
; Greek
  $DESCRIPTION =      "Sample driver description"
  $HELP =             "Sample help text information"
;DrIvEr DeScRiPtIoN EnD
```

General Syntax

This section describes the general syntax of a driver installation information file.

1. **You can add comment lines by starting the line with a semicolon (;). The parser ignores everything on the rest of that line. A semicolon can be preceded by white space (tabs or space characters). A comment cannot exist on the same line as a declaration. For example:**

   ```
   ; installation file for driver: NE2000.LAN
   ```

2. **Items in angle brackets indicate something that you must supply. The supplied item describes that aspect of the driver. For example:**

```
File: <filename>
File: NE2000.LAN
```

3. **Items in square brackets ([ ]) are optional.**

4. **Items separated by (or) indicate alternates. For example:**

```
THIS or THAT
```

5. **Labels are words that are followed by a colon. Labels are not case-sensitive. For example, the label *File:* is the same as *FILE:*.**

6. **Double quotes (" ") , single quotes (` ') , or whitespace can surround text strings.**

   a. **Strings without quotes must not contain white space, single or double quote characters, double-byte characters, or the following reserved characters: = { } ( ) , : - ; < > !**

   b. **Strings surrounded by single quotes can contain white space, single or double quote characters, double-byte characters, or the following reserved characters: = { } ( ) , : - ; < > !**

   c. **Strings surrounded by double quotes are treated as single-quoted strings. However, these strings are text strings that can be translated to other languages. For example:**

```
"The driver defaults to using ..."
`Novell NE2000'
 ISA
```

A single quote character can appear in a double- or single- quoted string if it is preceded by a backslash (\'). However, a double quote character cannot appear in a double-quoted string even if it is preceded by a backslash.

A backslash is represented within a single- or double-quoted string as (\\).

The newline and tab characters, \n and \t, can exist within single- or double-quoted strings. The parser changes the \n and \t to the appropriate characters.

7. **Keywords and labels must not be enclosed in quotes. Therefore, labels cannot contain white space, single or double quote characters, double-byte characters, or reserved characters.**

8. **Help text within double quotes can be longer than one line. When a**

quoted string spans more than one line, all characters from the last non-white space on one line to the first non-white space on the next line will be replaced with a single space character.

You can include a new line character in a quoted string by using \n. The parser will replace the \n by a CR-LF combination. \t indicates a tab.

Newline characters and tabs may only appear in help text and output format strings. For example:

```
"The ISADISK driver can be loaded twice. When
loaded more than once, the driver loads
reentrantly.\n\n

The default settings are the standard values
for an internal controller."
```

Language Translation

The installation information file's language section allows the translation of text strings to different languages. These text strings can include help messages and prompts. Language translation is optional and is not necessary for drivers that only operate in a single language.

If you implement the language section, any translatable text strings required in the driver descriptions use $<*string_name*> variables instead of the actual text. Each language block then contains the string name and the corresponding text in the respective language.

The *Language* and *DLanguage* labels identify a block of text string translations for either a particular language or the default language. If the *Language* label exists, the *DLanguage* label must also exist and must be the first language label. If only one language label exists in the file, it must be the *DLanguage* label.

When the installation utility encounters a $<*string_name*> variable, it searches for a definition of that string in the language block that corresponds to the installation/configuration utility's current language. If the string is not defined in the utility's current language, the utility searches string definitions in the default language block.

If a quoted string follows a $<*string_name*> with no intervening white space, and if the string definition is not found, the installation utility uses the quoted string as the string definition. This feature allows you to specify a default string (typically in English) if the string definition is not found in the language sections. If the installation/configuration

utility has already found a definition for the string, it will ignore the adjacent quoted string. For example:

```
$DESCRIPTION "Novell NE2000 Driver"
```

Finally, if the utility cannot find a string definition in any of the above mentioned forms, it will use the string name itself as the string text.

Language ID

A number, or language ID, identifies each language block. The language IDs that are currently assigned are listed below:

Canadian French	**0**
Chinese	**1**
Danish	**2**
Dutch	**3**
English	**4**
Finnish	**5**
French	**6**
German	**7**
Italian	**8**
Japanese	**9**
Korean	**10**
Norwegian	**11**
Portuguese	**12**
Russian	**13**
Spanish	**14**
Swedish	**15**

The NetWare operating system and utilities will be translated into German, Japanese, French, Spanish, and Italian. You can choose to provide text string translations for all, some, or none of the languages available. A sample installation information file with a Spanish language block is illustrated below:

Example

```
;DrIvEr DeScRiPtIoN

Version: 1.00
SyntaxVersion: 1.00
```

```
  Driver SAMPLE
  {
       Description:          $DRIVER_DESCRIPTION
       Help:              $DRIVER_HELP
 }

  DLanguage: 4
  ; Default English

    $DRIVER_DESCRIPTION    = "Place the driver description here"
    $DRIVER_HELP           = "Place the help information here"

  Language: 14
  ; Spanish

    $DRIVER_DESCRIPTION    = "Poner la descripcion del driver aqui"
    $DRIVER_HELP           = "Poner la informacion de ayuda aqui"

;DrIvEr DeScRiPtIoN EnD
```

Driver Section

The format of an information file's *Driver* description is shown below. The driver description contains two sections. The first section contains information the installation program uses to decide (with input from the user) if and how to install this driver. This section includes everything from the beginning of the section through the *Timeout* line. The second section contains the parameters that the user can change or select in order to configure the installed driver.

```
Driver <driver description name> <dependency
expression>
{
  Description:          "<text description>"
  Help:                 "<multi-line help text>"
  ParameterVersion:     <x.xx>
  ProductID:            <list of product ID strings>
  CProg:              <(server-specific) NLM name>
  Path:                 <path on media>
  File:                 <file name on media>
  OFiles:               <other associated files>
  Timeout:              <decimal timeout value in
                    seconds>

  PROMPT or LIST or FRAME
  {
 <see parameter section>
   }
}
```

All labels in the driver description are optional. You need only include the labels required for the particular driver being described. However we highly recommend that you define the *Description* and *Help* labels, to make installation and configuration easier for inexperienced users.

All labels and keywords can occur only once in a description. The exceptions to this are the *PROMPT, LIST, or FRAME* keywords and the *Parameter* definitions, which can occur as many times as you desire. The parameters are described later in this supplement. The following code fragment illustrates a sample driver description.

The following section describes the various parts of the above example.

Driver Description Name

Syntax:

Driver *<driver description name>*

Example:

Driver SAMPLE1

Each driver description has a case-sensitive string (*<driver description name>*) associated with it. This string is a logical name that uniquely identifies the driver description. (Remember that an information file can contain multiple driver descriptions.) This name must not be more than 32 characters, and cannot include white space, quotes (single or double), double-byte characters, or reserved characters (see ``General Syntax'').

After the user has installed and configured a driver, the *<installation filename>, <driver description name>* pair associates a description with the driver file. Therefore, each description name must be unique from all others within the same file.

Dependency Expression

Example:

```
if (BUS == MCA) OPTIONAL
else HIDDEN
```

A *Dependency* expression describes the state of a driver description. The dependency is either unconditionally or conditionally based on global parameters such as bus type (see ``Global Predefined Parameters''). A driver description has two states: optional and hidden. These states are defined as follows:

OPTIONAL **The driver description is displayed to the user, who can optionally select it.**

HIDDEN **The driver description is not displayed, and it is unavailable to the user.**

A dependency expression allows you to make descriptions invisible to the user if they are not applicable. (For example, a Micro Channel driver description would be hidden if the driver is being installed on an ISA machine). If no *Dependency* is declared, the driver description state defaults to optional.

For a more detailed description of the grammar and evaluation order in the dependency expressions see the ``Dependency Expressions'' section later in this supplement.

Description

Example:

```
Description: "Novell ISADISK (ISA or EISA) Driver"
```

The description label is followed by a case-sensitive string (or symbol reference), which is typically enclosed in double quotes. This string is displayed to the user during installation and configuration. The quoted string, if present, can be a maximum of 60 characters long, and must *not* contain newline characters (either symbolic \n or explicit).

Note: **You can have multiple *Description* labels in a driver description section. Each description must also have a corresponding *Help* label following it. The installation utility displays each description and help, but loads the same driver.**

Help

Example:

```
Help: "This driver supports up to four NE1000
  network boards installed in ISA servers. Their
  settings must not conflict.\n\n You  can load
the driver for each board and for  each
additional frame type assigned to the  board
  (maximum 16 times). The driver loads
reentrantly, thus conserving memory.\n"
```

The *Help* label is followed by a case-sensitive string, usually enclosed in double quotes, that the user could optionally have decided to display during installation and configuration. This string contains additional information or cautions that a user might need to know about the driver. The help can be a maximum of 1,500 characters long and can

contain newline characters (either symbolic \n or explicit). All explicit newline characters and adjacent whitespace are replaced with a single space. All symbolic new line characters are replaced with a CR-LF combination. \t represents a tab.

See also the note under the *Description* label in the previous section.

ParameterVersion

```
Example:
ParameterVersion: 1.00
```

The *ParameterVersion* refers to the version number of the driver parameters, (for example, the allowable command-line parameters in the case of a server driver). The *ParameterVersion* number should change *only* when the parameter interface changes, not when the installation file is modified. The installation file *Version* number is used to track file modifications.

ProductID

Examples:

```
; ProductID for Novell NE3200
   ProductID: NVL0701
; ProductIDs for IBM Token Ring and Token Ring 16/4
   ProductID: E000, E001
```

The *ProductID* label specifies unique identification string(s) assigned to the product. One or more comma-separated strings allow one driver to support several boards with different product IDs. For the Micro Channel Architecture, the string is the file name for the product's .ADF file, and its value is stored in the Micro Channel slot product ID POS registers. For the EISA architecture, the string is used as the name of the product's .CFG file, and is stored (in encoded form) in the EISA slot manufacturer ID and product ID registers.

Path

Example:
Path: \DRIVERS\LAN

If the *Path* label is present, the installation utility searches for the driver file in the directory path that the label indicates. This directory path resides on the distribution medium (in other words, on a floppy disk or a CD-ROM disk). If the *Path* is not present, the installation/configuration utility searches the root directory of the medium. See also the *File* label description.

File

Example:
File: NE2000.LAN

If the *File* label is present, the installation utility on the distribution medium looks for a driver file with the indicated name. The driver *Path* and *File* name are concatenated (using a `\' between them) to form the full directory specification on the installation medium. If the *File* label is not present, the installation utility uses the description file root name with a default extension. Server MLID files use a .LAN extension; server disk driver files use a .DSK extension.

OFiles (Other Associated Files)

Example:
OFiles: FIRMLOAD.COM, MONT400.BIN

If the *OFiles* label is present, a comma-separated list of filenames must also appear on the same line. When the primary installation/configuration utility copies the driver file, it will also copy these associated files. The *Path* string is concatenated to each of the listed files (using a `\' between them) to form the full directory specification for each file.

CProg (Configuration Program)

Example:
CProg: CSL.NLM

The *CProg* label specifies a configuration executable and contains the name of an NLM that performs the configuration.

Timeout

Example:
Timeout: 20

If the *Timeout* label is present, it must be followed by a decimal number. This decimal number indicates the maximum time in seconds that the installation utility waits before it determines that a driver failed to load and reports an error to the user. If this label is not specified, the maximum wait time defaults to 5 seconds.

Driver Parameters

Each of the driver's configurable parameters must be defined in the

driver description by using one of the parameter types detailed in this section.

Parameter specifications define the configurable parameters that the driver needs. A parameter specification includes several components: the parameter values, the presentation to the user, and the output format. The output format controls how the server driver information will be written to the command line.

Three types of parameters are allowed; two are general parameters, and one is a special purpose parameter.

The two general parameter types are *PROMPT* and *LIST*. They can occur more than once in a driver description. They occur once for every user-configurable driver parameter. Both parameters contain fields for a parameter description and help text, dependency expressions, and output format specification. You can also specify a default value for the parameter, as well as permissible values from which the user can choose.

***FRAME,* the special purpose parameter type, can be declared only once within a single driver description. This parameter defines the frame types that are supported by the NetWare server MLID. As with the general parameters, the *FRAME* parameter allows a description, help, and a dependency expression. This parameter uses a default method for input and output.**

The general definitions that apply to all three parameters are described below. The following pages then provide the specific syntax for each parameter.

General Parameter Definitions

The parameter name is a case-sensitive string of from 1 to 16 characters. The parameter name is not displayed on the monitor. It is used only to reference another parameter's value in a dependency expression and to allow the installation/configuration utility to distinguish between driver descriptions. A parameter name can occur only once within a single description.

All configurable parameters can have a default value of UNDEFINED. This value indicates that no initial value is specified. If the parameter value remains UNDEFINED, the driver can determine the appropriate values automatically.

A parameter can exist in one of three states: HIDDEN, REQUIRED, or OPTIONAL. The parameter state affects user input and output as follows:

HIDDEN **Indicates that the parameter is invisible to the user.**

REQUIRED **Allows the installation program to determine which parameters are required by the driver at load time. The parameter is displayed, and a valid value must be specified for the parameter (either a default value or a value entered by the user). Output is always generated.**

For example, if a driver has a REQUIRED port parameter, the user may not exit the parameter form until a valid value is selected for the parameter. The string, ``PORT=*xxxx*'' will always be generated on the command line after the ``LOAD <driver>...'' string.

***PROMPT, LIST,* or *FRAME* parameters that are specified as REQUIRED, but have only one valid choice (the default value), have the following unique features: (1) the parameter is not displayed to the user, because the user has no choice to make, and (2) the parameter will generate output. This feature creates an ``invisible'' parameter that generates output.**

OPTIONAL **Signifies the parameters that are allowed but are not required by the driver. The parameter is displayed to the user, but no input is required from the user.**

If the parameter has no default value specified, the user may leave it unspecified. If the value of a parameter is not specified, or if a valid default is deleted by the user, no output is generated (in other words, no output for the parameter is displayed on the command line).

If the parameter has a default value specified, and the user accepts the default, no output will be generated for the parameter. Otherwise, if the user changes a parameter to a defined value different from the default, output will be generated. A default value should be specified for an optional parameter if, and only if, the driver will default the parameter to that value.

A dependency expression decides the state of a parameter under various conditions. A parameter state can be specified as

unconditionally OPTIONAL or REQUIRED. It can also be specified as conditionally OPTIONAL, REQUIRED, or HIDDEN and depending on the value of other parameters. If a parameter has no dependency, its state defaults to unconditionally OPTIONAL. Refer to the section ``Dependency Expressions'' for a more detailed description of parameter states with dependency expressions.

The *PROMPT* Parameter. The format of the *PROMPT* parameter is shown below. *PROMPT* obtains user input for a configurable parameter. The parameter can be a custom parameter or a local predefined parameter (see ``Local Predefined Parameters'').

The installation/configuration utility uses the specified *Description* string and *Default* value (if any) to prompt the user to enter a value for the parameter. The user can then accept the default value or choose another value from a specified set. The following code fragment illustrates the use of a *PROMPT* parameter. The *Description* and *Type* fields shown below are required; all other fields shown are optional.

Example

```
Syntax:
PROMPT <parameter_name> <dependency expression>
{
  Description:     "<description text>"
  Help:            "<multi-line help text>"
  Type:         STRING (max_chars) or
                   HEX (max_digits) or
                   DECIMAL (max_digits)

  Values:          <minimum value> - <maximum value> or
                <value 1>, <value 2>, ... <value n>

  Default:         <default value> or UNDEFINED

  ReservedLength: <hexadecimal length of values
                reserved or <name>>

  OutputFormat:       `<any string with a %s>'
}
```

The following section describes the different portions of the above example.

Dependency Expression. *PROMPT* parameters can be assigned a state of REQUIRED, OPTIONAL, or HIDDEN. You can use a conditional dependency expression to determine the parameter state. You can also use *PROMPT* parameters in the dependency expressions for other

parameters. When used in dependency expressions, the *PROMPT* parameter value is the value selected by the user (or UNDEFINED if no value was specified). Refer to ``Dependency Expressions'' for a more detailed description of the dependency usage.

Description. The *<description text>* is the prompt for the user configurable parameter. The description string can be a maximum of 40 bytes.

Help. The *Help* text can be longer than one line and can be a maximum of 1,500 bytes.

Type. Type specifies whether a value is interpreted as string, hexadecimal, or decimal. You can optionally specify the maximum number of characters or digits for that value. (Parameter values can have a maximum of 35 characters or digits.) If the maximum length, *<max_chars>* or *<max_digits>,* is not specified, it defaults to the maximum element size in the list of *Values* (described below). If no values are specified, 8 characters are used for parameters having predefined names (see ``Local Predefined Parameters''), and 35 characters are used for parameters without predefined names.

Example 1 below would allow the user to enter up to 35 characters and generate the output as indicated. Example 2 would allow only 10 characters.

```
Example 1:

  PROMPT param2
  {
       Description: "A string parameter"
       Type: STRING
       Default: `my_string'
       OutputFormat: `String=%s'
  }

Example 2:

  PROMPT param2
  {
       Description: "A string parameter"
       Type: STRING (10)
       Default: `my_string'
       OutputFormat: `String=%s'
  }
```

Values. This field indicates the allowable values for the parameter. The values are displayed on the console as the user highlights the

parameter field. The values can be specified by using a range of values or by using a comma-separated list of values. The range or list of values to be displayed must be less than 70 characters long.

Default. The default value is optional and, if used, is displayed along with the description string as part of the parameter prompt. The default value must be of the specified *Type* and must be an element indicated in the *Values* range or list. The default value can also be UNDEFINED. An absent *Default* label is identical in function to a default value of UNDEFINED.

ReservedLength. The *ReservedLength* label is generally ignored, even if it is present. The exception to this is the cases of the *PORTx* and *MEMx* reserved parameters. In these cases, the server environment requires these labels (see ``Local Predefined Parameters''). If this label is present, it must contain either a single hexadecimal constant or the name of another parameter whose value (when set) will be used as the reserved length of this parameter. The *ReservedLength* value has the type specified by the *Type* label.

OutputFormat. The *OutputFormat* string describes the way the output is to be generated from the final parameter value. The output is displayed on the server monitor's commandline. The format string can contain a maximum of one %s. The output string is created by replacing the %s with the parameter value. Output is generated according to the rules described in the section ``General Parameter Definitions.''

The example below shows a sample PROMPT parameter block using the local predefined parameter, INT, and the resulting screen displayed in the installation utility.

Note: **The *Description, Help,* and *Type* fields shown below are included to illustrate their use in the PROMPT example. For the local predefined INT parameter, these fields have default values and are not usually required.**

Example:

```
PROMPT INT REQUIRED
{
  Description:       "Interrupt"
  Help:              "Select the primary interrupt number."
  Type:  HEX(1)
  Values:  2, 3, 5, 7
  Default: 3
```

```
  OutputFormat: `INT=%s'
}
```

Resulting Screens:

Driver NE2000 parameters

Supported values: 2,3,5,7 Default value: 3 **Select the primary interrupt number.**

The *LIST* Parameter. The format of the *LIST* parameter is shown below. *LIST* obtains user input for a configurable parameter. *LIST* is similar to *PROMPT*, with the exception that the user selects an option for the parameter from a menu of valid choices.

The installation utility uses the parameter *Description* and the *Default* choice description (if any) to prompt the user for a selection. The user can then accept the default choice or select another from the menu of choices for the parameter. The *Description* parameter and the *Choice* fields shown below are required; all other fields shown are optional.

```
Syntax:
LIST <parameter_name> <dependency expression>
{
  Description:        "<parameter description text>"
  Help:               "<multi-line help text>"
  CDescription:       "<choice #1 text>"
  Choice:             <choice #1 value> or UNDEFINED
  .
  .
  .
  CDescription:       "<choice #n text>"
  Choice:             <choice #n value> or UNDEFINED
  Default:            <1 to n> or UNDEFINED
  OutputFormat:       `<format string with a %s>'
}
```

The various parts of this example are described below:

Dependency Expression. *LIST* parameters can be assigned a state of REQUIRED, OPTIONAL, or HIDDEN. A conditional dependency expression can be used to determine the parameter state. *LIST* parameters can also be used in the dependency expressions for other parameters. When used in dependency expressions, the *LIST* parameter value is a decimal number indicating the index of the *Choice* selected by the user (or UNDEFINED if no choice was specified). Refer to ``Dependency Expressions'' for a more detailed description of the dependency usage.

Description. The *<parameter description text>* is the prompt for the user configurable parameter. The description string can be a maximum of 40 bytes.

Help. The *Help* text can be longer than one line and can be a maximum of 1,500 bytes.

Choice and CDescription. The installation utility uses *Choice* or *CDescription* to create a menu of valid choices for the parameter. The description text string is typically enclosed in double quotes (if language translation is supported), because the menu choices can usually be translated to different languages.

The installation utility uses the *Choice* field to build the command line entry (see the ``OutputFormat'' description below). Choice values can be any string, including the null string, or can be UNDEFINED. Ranges are *not* allowed. (For example, Choice: 1-50 is illegal.) Typically choice values will not be enclosed in double quotes, because this results in language specific command line or configuration file parameters.

If the *CDescription* is not provided for a particular *Choice*, the menu text will be the choice string itself. The number of pairs of choice descriptions pairs implies the number of choices. The maximum field width for any given choice description is 35 characters.

Default. The *Default* value is a decimal number indicating the index of the default choice. It must be in the range of 1 to the number of choices. The default value can also be UNDEFINED, which means that none of the choices are initially selected. An absent *Default* label is identical to a default value of UNDEFINED.

The default value is optional and, if used, the corresponding *CDescription* is displayed at the parameter prompt to indicate the default choice. If the default value is specified as UNDEFINED then ``(not specified)'' will be displayed.

OutputFormat. The *OutputFormat* label describes the way the output is to be generated from the selected parameter choice. The output is displayed on the server monitor's commandline. The format string can contain a maximum of one %s. The output string is created by replacing the %s with the selected *Choice* string. The output is generated according to the rules described in the section ``General Parameter Definitions.''

The example below shows a sample *LIST* parameter block and the resulting screens displayed in the installation utility.

Example

```
LIST Attach_Mode OPTIONAL
{
  Description:    "FDDI Station Attach Mode"

  Help:           "If there is a secondary board in
            your machine, you may wish to
          override the auto sense attachment.
   Select the correct mode from the
  list."

  CDescription: "Single Attach"
  Choice:  `1'

  CDescription: "Dual Attach"
  Choice:  `2'

  CDescription: "Auto Sense"
  Choice:       UNDEFINED

  Default:  3

  OutputFormat: `ATTACH_MODE=%s'
}
```

Special Purpose Parameter Definition

FRAME Parameter. The format of the *FRAME* parameter is shown below. *FRAME* allows the user to select the NetWare server MLID's default frame types.

The installation utility uses the *Description* parameter and the *Default* values to display a list of frame names. The user can add or delete frames from the list. In the server environment, a default logical name can also accompany each frame type. Only the *Choice* fields shown below are required; all other fields shown are optional.

Example

```
Syntax:
FRAME <parameter_name> <dependency expression>
{
  Description:        "<parameter description text>"
  Help:               "<multi-line help text>"
  CDescription:       "<frame #1 description text>"
  Choice:             <frame #1 type string>
    .
    .
    .
  CDescription:       "<frame #n description text>"
  Choice:             <frame #n type string>

  Default:            <1,...,n> or UNDEFINED
  OctetBitOrder:      <LSB or MSB>
}
```

The following section describes the various portions of the above example:

Dependency Expression. If the *FRAME* parameter state is OPTIONAL, the user does not need to indicate any values. If the parameter is REQUIRED, the user must select at least one frame type.

Only one *FRAME* parameter with multiple frame types can be visible (REQUIRED or OPTIONAL) to the user. Multiple *FRAME* parameters can be declared, but only one block can be active; all others must be HIDDEN. (If the parameter is HIDDEN, nothing will be presented to the user, and no output will be generated for that parameter.)

A *FRAME* parameter can also be used in dependency expressions for other parameters. When used in dependency expressions, the parameter's value is a nonzero decimal number indicating the number of frame types selected (or UNDEFINED if no frame types were specified). For a more detailed description of dependency usage see ``Dependency Expressions'' .

Description. The *<parameter description text>* is the frame type prompt. The description string can be a maximum of 40 bytes. If the

description is not present, the text will default to ``Frame Types.''

Help. The *Help* text can be more than one line long and can be a maximum of 1,500 bytes. If the help text is not present, the default Frame help text is displayed (see the ``Default Help Information'' table in the ``Local Predefined Parameters'' section later in this supplement).

Choice and CDescription. The *Choice Description (CDescription)* fields create the list of default frame types. The maximum field width for any given frame description is 35 characters. If *CDescription* is not provided for a particular *Choice*, the text displayed to the user will be the frame type string itself.

***Choice* values can be any string, but should be strings that are understood by the NetWare server MLID and the protocols that will be used. Each *Choice* can appear only once in the list. Typically frame types are not enclosed in double quotes, because this would result in language specific commandline or configuration file parameters.**

Default. The *Default* field contains a list of numbers corresponding to default frames, where 1 corresponds to the first frame type, 2 to the second, etc. The value may also be UNDEFINED, indicating that no default frame names are initially selected. An absent *Default* label is identical to a default value of UNDEFINED.

OutputFormat. The output for the *FRAME* parameter is implied (the OutputFormat label is not used). In the server environment, the load driver command will be reiterated at the command line for each frame type selected.

OctetBitOrder. This label is optional and, if used, should only be present for Token-Ring and PCNII networks. The *OctetBitOrder* field allows the user to specify whether network addresses are in canonical or noncanonical (LSB or MSB) formats (see the *ODI Specification Supplement: Canonical and Noncanonical Addressing*.) The value associated with this label will be the default value for all frame types.

The example below shows a sample FRAME parameter block and the resulting screen displayed in the installation utility.

Example (for Assembly Language NetWare server MLIDs)

```
FRAME FrameSelect
{
Help:              "The driver defaults to the 802.2
```

```
                frame type. You can optionally
              remove this frame type and/or
            add the 802.3, 802.2 SNAP, and/or
          Ethernet II frame types."

  CDescription:       "802.3"
  Choice:             `Ethernet_802.3'

  CDescription:       "802.2"
  Choice:             `Ethernet_802.2'

  CDescription:       "802.2 SNAP"
  Choice:             `Ethernet_SNAP'

  CDescription:       "Ethernet II"
  Choice:             `Ethernet_II'

  Default:               2
}
```

Local Predefined Parameters

Predefined *PROMPT* Parameter

Some local parameters are standard. Therefore, these local parameters can have more specific meanings than the general parameter definitions mentioned in the previous sections. The following table lists the predefined *PROMPT* parameter names:

Predefined *PROMPT* Parameters

Name	Parameter Type	Meaning

INT or INT1		
INT2	**PROMPT**	**Primary interrupt**
PORT or PORT1	**PROMPT**	**Second interrupt**
PORT2	**PROMPT**	**Primary I/O port**
MEM or MEM1	**PROMPT**	**Second I/O port**
MEM2	**PROMPT**	**Primary memory address**
DMA or DMA1	**PROMPT**	**Second memory address**
DMA2	**PROMPT**	**Primary DMA address**
SLOT	**PROMPT**	**Second DMA address**
NODE	**PROMPT**	**Machine slot number**
RETRIES	**PROMPT**	**Node address**
CHANNEL	**PROMPT**	**Number of retries**
	PROMPT	**Channel number for adapters that use multiple NIC controllers**

If a *PROMPT* parameter name is one of the predefined names listed above, all of the labels are optional, and will be defaulted if they are not specified. If a dependency expression is not declared, the parameter state will default to OPTIONAL. The range or list of values that the user can enter excludes ranges or values that other drivers are already using.

If one or more of the fields are not specified for the local predefined parameters, the installation/configuration utility generates the following defaults:

Predefined Parameter Defaults

Parameter Field	**Default Information**

Description:	**(INT)``Interrupt number''** **(INT2)``Secondary interrupt number''** **(PORT)``Port value''** **(PORT2)``Secondary port value''** **(MEM)``Memory address''** **(MEM2)``Secondary memory address''** **(DMA)``DMA value''** **(DMA2)``Secondary DMA value''** **(SLOT)``Slot Number''** **(NODE)``Node Address''** **(RETRIES)``Number of Retries''**
Help:	**The default help information is listed following this table.**
Type:	**DECIMAL(8) for SLOT and RETRIES** **HEX(12) for NODE** **HEX(1) for INT and INT2** **HEX(8) for all others**
Values:	**0-99999999 for SLOT and RETRIES** **0-FFFFFFFFFFFF for NODE** **0-F for INT and INT2** **0-FFFFFFFF for all others**
Default:	**UNDEFINED**
ReservedLength:	**Not defaulted. This field must be specified for a *PORTx* or *MEMx* parameter in the server environment.**

OutputFormat:

(INT)`INT=%s'
(INT2)`INT1=%s'
(PORT)`PORT=%s'
(PORT2)`PORT1=%s'
(MEM) `MEM=%s'
(MEM2) `MEM1=%s'
(DMA) `DMA=%s'
(DMA2) `DMA1=%s'
(SLOT) `SLOT=%s'
(NODE)`NODE=%s'
(RETRIES)`RETRIES=%s'

Default Help Information

Name	**Text**
INT **INT1**	**``\nSelect the interrupt level that corresponds to the interrupt setting on the board or other device.\n\n The interrupt setting must be unique (one not used by another device in the machine)."**
INT2	**``\nSelect the interrupt level that corresponds to the second interrupt setting on the board or other device.\n\n The interrupt setting must be unique (one not used by another device in the machine)."**
PORT **PORT1**	**``\nSelect the port value (base I/O address) that corresponds to the port address setting on the board or other device.\n\n Make sure the block of I/O addresses does not overlap the addresses of another device in the machine."**

PORT2	**``\nSelect the port value (base I/O address) that corresponds to the second port address setting on the board or other device.\n\n Make sure the second block of I/O addresses does not overlap the addresses of another device in the machine."**
MEM **MEM1**	**``\nSelect the memory address that corresponds to the memory setting on the board or other device.\n\n Make sure the block of memory addresses does not overlap the addresses of another device in the machine."**
MEM2	**``\nSelect the memory address that corresponds to the second memory setting on the board.\n\n Make sure the block of memory addresses does not overlap the addresses of another device in the machine."**
DMA **DMA1**	**``\nSelect the DMA channel that corresponds to the DMA setting on the board or other device.\n\n Make sure the DMA (Direct Memory Access) channel does not conflict with that of another device in the machine."**
DMA2	**``\nSelect the DMA channel that corresponds to the second DMA setting on the board.\n\n Make sure the DMA (Direct Memory Access) channel does not conflict with that of another device in the machine."**
SLOT	**``\nSelect the slot number that corresponds to the expansion slot where the board or other device is installed."**
NODE	**``\nDo not change this address unless you are prepared to administer local addresses according to the IEEE 802.2 specifications.\n\n The driver defaults to the node address on the board."**

RETRIES	**``\nThis number specifies the maximum number of times the driver will be instructed to retry a failed packet transmission."**
FRAME	**``\nSelect the frame type used by the protocol your network requires.\n\n If you select a frame type other than the default, configure both client and server to use the same frame type."**

The *INT* Predefined Parameter

The following examples show how to use the use the predefined parameter *INT*.

```
Example 1:
  PROMPT INT
  {
  }
Example 2:
  PROMPT INT
  {
       Type:        DEC (2)
       Values:         2, 3, 4, 5, 10
       Default:     3
  }
```

In Example 1, the parameter description, output format, type, and type length default as follows:

```
Description:       "Interrupt number"
OutputFormat:      `INT=%s'
Type:           HEX (1)
Values:            0-F
Default:        UNDEFINED
```

The help information displayed for this parameter would be:

```
``Select the interrupt level that corresponds to the
interrupt setting on the board or other device.
The interrupt setting must be unique (one not used by
another device in the machine).''
```

In Example 1, the installation/configuration utility will not allow the user to enter an interrupt value that is already taken, even though the taken values are not specified in the help text.

However, in Example 2, assume that another NetWare server MLID is already using interrupt 3. The parameter description and output format default as follows:

```
Description:       "Interrupt number"
OutputFormat:      `INT=%s'
```

The help information displayed for this parameter would be:

```
Permissible values:      2, 4, 5, 10
Default value:           3 (not-selectable)

Select the interrupt level...
```

Please note in example 2 that the explicitly declared field Type: DEC

(2) allows 2 digits to be defined for interrupt 10. Also note that if any parameter field is explicitly declared do not use the default information as defined in Tables 4 and 5.

The *PORTx* and *MEMx* Predefined Parameters

In the case of *PORT*x and *MEM*x, the *ReservedLength* is a required label and must be present as part of the *PROMPT* parameter. *ReservedLength* determines whether the specified group of port or memory addresses are available, and prevents the user from entering values that are taken. *ReservedLength* must contain either a single hexadecimal constant or the name of another parameter whose value (when set) will be used as the reserved length of this parameter. If the parameter is *PORTx*, the reserved length represents a range of port values in bytes. If the parameter is *MEMx*, the reserved length represents a range of memory addresses in paragraphs (groups of 16 addresses).

Global Predefined Parameters

The following table lists the globally predefined parameters. These parameters are never displayed to the user (although in some cases the user will be prompted by the installation utility for the information needed to create them). They exist only to allow driver description and parameter dependency expressions to reference them. This makes descriptions' and parameters' states conditional upon the value of these global parameters. You can write dependencies assuming that these values will always exist and that they will never be UNDEFINED.

Globally Predefined Parameters

Name	Parameter Type	Possible Values

BUS	**PROMPT, STRING**	**`ISA'** **`MCA'** **`EISA'** **PCMCIA** **PCI**
GT_16	**PROMPT, STRING**	**`TRUE'** **`FALSE'**

The *BUS* Parameter

***BUS* indicates which bus architecture the installation/configuration utility is working with. (The is the bus architecture of the machine for which the command line information is being created.)**

The *GT_16* Parameter

If the *GT_16* parameter value is TRUE, the machine has more than 16MB of memory available for the driver to use.

Dependency Expressions

Dependency Expression Syntax

As mentioned previously, a dependency expression can appear in the context of a driver description (for example, before the Driver { }), or in the context of a parameter (for example, before the parameter { }).

Used in the context of a driver description, the dependency specifies the conditions under which the entire driver description is HIDDEN (invisible, inaccessible) or OPTIONAL (visible, selectable) to the user.

Used in the context of a parameter, the dependency specifies the conditions under which the parameter is HIDDEN (invisible, no input, no output), REQUIRED (visible, input required, output required), or OPTIONAL (visible, input optional, output optional).

A dependency has the following syntax:

```
<dependency expression>     <- <state> or
                            if (expression) <state>
                            [else if (expression) <state>...]
                            else <state>

<state>                     <- OPTIONAL
```

```
                                    HIDDEN
                                    REQUIRED (parameter context
only)

<expression>              <-        <log-expr> or

<expression><log-op><log-expr>
<log-op>                  <-        OR  or  AND

<log-expr>                    <-        <rel-expr> or NOT <rel-
expr>

<rel-expr>                        <-        (<expression>) or
                                  <name><rel-op><name> or
                                  <name><rel-op><constant>
<rel-op>                          <-        ==   !=   <   >   <=   >=
```

<name>. *Name* can be either a global predefined parameter or a local parameter that precedes this parameter in the driver description. If the parameter named is a *PROMPT* parameter, that value depends on the *PROMPT*'s Type label. If the parameter named is a *LIST* or *FRAME* type, its value is a decimal number. String comparisons are *not* case-sensitive.

If the dependency expression is in the context of a driver description, *Name* refers *only* to global predefined parameters (see ``Global Predefined Parameters'').

If the dependency expression is in the context of a parameter, *Name* refers to either a global predefined parameter or to a local parameter that precedes this parameter and is in the same driver description.

<constant>. A *constant* may be either numeric or string-valued. Its type is assumed to be the type of the parameter to which it is being compared. All cross-type comparisons between strings and numbers are flagged as syntax errors. The typeless constant value, UNDEFINED, is used for comparing against parameters that do not have a defined value.

The following is an example of a conditional dependency statement:

Example:

```
PROMPT parameter2
if (parameter1 == 5 AND BUS == MCA)
  OPTIONAL
else if (parameter1 != UNDEFINED AND parameter1 != 5)
  REQUIRED
```

```
else
  HIDDEN
{
  .
  .
  .
}
```

General Dependency Expression Syntax Rules

With the exception of the global predefined parameters, all names used in dependency expressions must be defined previously in the driver description. No forward references to a name are allowed. Any attempt to forward-reference a name will be flagged as an error and the driver description will be discarded.

No circular references to a name are allowed (in other words, a name cannot directly or indirectly depend upon itself).

In dependency expressions, any reference to a parameter name whose state is HIDDEN, or whose value is not specified (this could be due to an OPTIONAL parameter whose default value is UNDEFINED, or an OPTIONAL parameter whose default value was defined, but the user deleted it), will return a value of UNDEFINED for that parameter.

A REQUIRED parameter must have a valid (defined) value before the user can exit the form. As a result, you can write dependency expressions assuming that a REQUIRED parameter will always have a defined value and will never be UNDEFINED.

Evaluation of Dependency Expressions

Terms with == or != expressions that reference a parameter with an UNDEFINED value yield a valid result. All other relational operators result in an error for the term. Explicitly, the following expressions are valid if *name* has an UNDEFINED *value*.

```
name == value
name != value
```

The following expressions will result in a term evaluation error if *name* is UNDEFINED.

```
name >= value
name <= value
name > value
name < value
```

This also applies to expressions comparing two parameters (for example, *name1* >= *name2*).

The installation/configuration utility resolves dependency evaluation errors, if they occur, by forcing the state of the parameter or driver description to OPTIONAL and reporting the error to the user before he or she exits the parameter form. (This error is nonfatal, and the driver could still work if the resultant output is reasonable.)

In order to prevent term evaluation errors from resulting in evaluation errors for the entire dependency, you should account for the UNDEFINED value when you write descriptions (either explicitly or implicitly). A dependency expression that explicitly handles an UNDEFINED value has a comparison to UNDEFINED should appear *prior* to the term that could result in an error. For example:

```
if (param1 != UNDEFINED AND param1 > 30) REQUIRED
else HIDDEN
```

The above expression will *not* result in an dependency evaluation error if param1 has an UNDEFINED value.

The following example illustrates an implicit comparison that takes UNDEFINED into account:

```
if (param2 == 3) REQUIRED
else HIDDEN
```

The above expression will result in a HIDDEN state if param2 has an UNDEFINED value.

Example NetWare Server MLID

```
;DrIvEr DeScRiPtIoN
SyntaxVersion: 1.00

   Driver PCN2
 {
        Description: "Novell PCN2 (ISA or MCA) Driver"

      Help:             "You can use this driver in an ISA (AT bus), EISA,
                        or a Micro Channel file server. You can have a
                        maximum of two PCNII network boards in your file
                        server."

         File:             PCN2.LAN
      ParameterVersion:    1.00
```

```
  PROMPT PORT

   if (BUS != MCA) REQUIRED
      else HIDDEN
 {
      Values:               620, 628
      Default:                 620
      ReservedLength:          8
 }
      PROMPT SLOT

   if (BUS == MCA) REQUIRED
   else HIDDEN
 {
      Values:               1-8
 }

PROMPT NODE
 {
 }
      FRAME FrameSelect
 {
         Help:              "The driver defaults to using the PCNII 802.2 frame
                         type. You can optionally remove this frame type
                         and/or add the PCNII 802.2 SNAP frame type."

  CDescription:             "PCNII 802.2"
  Choice:                   `IBM_PCN2_802.2'

  CDescription:             "PCNII 802.2 SNAP"
  Choice:                   `IBM_PCN2_SNAP'
  Default:                  1
 OctetBitOrder:             LSB
 }
 }
;DrIvEr DeScRiPtIoN EnD
```

Driver Installation Information Template

The following is an installation information file template for NetWare server MLIDs. This template can also be used for server disk drivers by deleting LAN-specific parameters such as node, frame, and protocol.

All translatable strings in the following template are surrounded by double quotes, as in ``*<translatable string>*''. If you want to add new strings, follow the convention to surround *only* strings that should be translated in double quotes. All other strings should either not have quotes or should be surrounded by single quotes. Write and test your information file and check it with (but not concatenated to) the driver.

Replace all strings shown in the *<xxxx>* format with the appropriate string. You can delete any description line that you do not need. You can also add any additional custom parameters that you need.

```
;DrIvEr DeScRiPtIoN
VER: 1.00
SYN: 1.00
; Place introductory comments here.
; Keep comments and white space to a minimum.

DR <Driver Description Name> <Dependency Expression>
{
   DES:  "<text description>"
   HELP: "<multi-line help text>"
   PAR:     X.xx
  PROD:  <product ID string>
   CP:      <configuration NLM name>
  PATH:  <path on media>
   FILE: <file name on media>
   OF:      <other assoc. files>
   TIME: <driver load timeout value>

; You will need only some of the following for any given driver description.
; Delete those not needed and edit those which you do need to correctly      ;
describe your adapter and driver. You can add custom parameters to describe ;
driver parameters not covered here. Most likely, one or more of the        ;
parameters (INT, PORT, MEM, etc.) will be indicated as REQUIRED".

PR INT
 {
      VAL: 3,5,7,9
      DEF: 9
 }
 PR PORT
 {
```

```
      VAL: 300,310,320
      DEF: 300
      RES: 8
; (continued)
 ; ReservedLength for port specifies a range in single value increments
 }
   PR MEM
 {
      VAL:     C000,C800,D000,D800
      DEF:     C000
      RES:     800
 ; ReservedLength for memory specifies a range in paragraphs
 }
   PR DMA
 {
      VAL: 1,3,5,7
      DEF: 3
 }
   PR SLOT
 {
      VAL: 1-8
 }
   PR NODE
 {
 }
   FR FrameSelect
 {
      HELP:       "The defaults are set to 802.2 and 802.3 frame types.\n\n

                 We strongly recommend that you select at least 802.2.
                 For existing networks, select both 802.2 and 802.3"
  CD:            "802.3"
  CH:               `Ethernet_802.3'

  CD:               "802.2"
  CH:               `Ethernet_802.2'
  CD:               "802.2 SNAP"
  CH:               `Ethernet_SNAP'
  CD:            "Ethernet II"
 CH:             `Ethernet_II'
 DEF: 1,2
 ; For Ethernet server drivers, set the default to 802.2 and 802.3.
 ; For all other drivers, set the default to whatever the default
 ; is in the driver, and change the help text accordingly.
 }
}

;DrIvEr DeScRiPtIoN EnD
```

The 16-Bit DOS Client Installation Information (INS) File

If your 16-bit DOS ODI LAN driver will be installed with the installation utility, the utility must know how to set up the configuration file (NET.CFG). This means the utility must know each driver's parameter options and which choices the user must make. The driver installation information (.INS) file and the DRIVER.LST file provide installation utilities with this required information.

Each driver has only one .INS file even though the driver usually supports many boards. If the installation utility does not find an INS file for a driver, the utility does not create a NET.CFG file entry for the driver. The installation program uses the .INS file to prompt the user for the parameter options and values necessary to generate a NET.CFG file. Sample INS files are included at the end of this supplement.

Each driver distribution diskette may include one DRIVER.LST file. This file provides the installation program with a quick directory of all ODI drivers found on the distribution diskette.

General Rules

Your INS file must conform to the following rules and also to the conventions used in the examples at the end of this supplement.

- **The INS file must not contain blank lines or tabs. Space characters are not permitted between fields unless otherwise specified.**
- **Items shown in angle brackets indicate that the user must supply information describing that aspect of the driver.**

  example: **<DriverName>**
  NE2000.COM
- **Items shown in square brackets ([ ) and ( ]) are optional.**

File Syntax

The format of the INS file for 16-bit DOS ODI drivers is shown below:

```
InS_StArT
<DriverName>
<Version>[,<AssociatedFileList>]
^<Board1 Description>
^<Board2 Description>
    $
    $
    $
[?<Help Text>]
<Parameter1 Definition>
<Parameter2 Definition>
    $
    $
    $
InS_EnD
```

Signature Lines

Note the initial and final lines:

```
InS_StArT
$
$
$
InS_EnD
```

These lines illustrate the following crucial points:

- **The InS_StArt and InS_EnD signature lines shown above bracket the driver installation information.**
- **The body of the INS file is preceded by the ``InS_StArT'' keyword and is terminated by the ``InS_EnD'' keyword.**

Driver Name

The driver filename, including the .COM or .EXE extension, is the first line of the installation information. Installation uses the driver filename to generate the NET.CFG Link Driver <DriverName> command.

Version

The *version* field indicates the version of the INS specification to which the INS file is written. The version field enables installation utilities to properly handle any future changes to the specification. The version number format is `X.X.' For this specification, the version is 1.1.

Associated File List

The version number may be optionally followed by a list of associated files separated by commas. The user must copy these files, along with the driver, to the area where the user is installing the client pieces. Firmware BIN files and special configuration or diagnostic utilities for the adapter or driver are examples of associated files.

Board Description

The *Board Description(s)* provides a list of adapters and related information that the DRIVER.LST file uses. (DRIVER.LST is described later in this appendix.) More than one board can be listed if a single driver will work with a given adapter and its clones. Place multiple board descriptions on separate lines in the DRIVER.LST file and only use them when the hardware options are identical for all the boards. The user generally knows what board he/she has installed, but not necessarily what driver he/she should use with that board.

Each *Board Description* is preceded by a caret (^) and has the following format:

syntax: ^<BoardName,DriverName,BusCode[,ProductID]>

example: ^Novell NE2000,NE2000.COM,IEO

- ***BoardName* is the full name of the network adapter. This field can contain spaces and can be a maximum of 48 characters long.**
- ***DriverName* is the filename (including the extension) of the board's driver. This field can be a maximum of 13 characters long.**
- ***BusCode* is a six character code depicting the bus type(s) supported for the adapter. The code is described below:**

```
1st character   I (alpha)    =   supports ISA adapters
    0 (numeric)     =   does not support ISA adapters
2nd character  E (alpha)   =   supports EISA adapters
    0 (numeric)     =   does not support EISA adapters
3rd character   M (alpha)   =   supports MCA adapters
    0 (numeric)     =   does not support MCA adapters
4th  character  A (alpha)   =   supports PCMCIA adapters
    0 (numeric)     =   does not support PCMCIA adapters
5th  character  P (alpha)   =   supports PCI adapters
    0 (numeric)     =   does not support PCI adapters
6th  character  V (alpha)   =   supports VESA Local Bus
adapters
    0 (numeric)     =   does not support VESA Local Bus
```

adapters

Note: Because all ISA boards work in EISA machines, `I' and `E' should both be used for ISA boards. This enables ISA boards installed in an EISA bus to appear in the installation utility. Keep in mind that ISA boards installed in EISA machines retain ISA functionality and features.

- **The *ProductID* field is optional and only applies to MCA and EISA boards. This field contains the ID string that is stored in the POS registers of MicroChannel and in configuration registers of EISA machines. The installation utility uses this ID string to select the appropriate driver for the board found in the machine. This field is left blank for ISA boards.**

Help Text

You can supply one line of optional help text after the board description lines. The help text is preceded with a question mark (?), can be up to 256 bytes in length (not including the question mark), and can contain spaces. The installation program automatically word wraps the help text. The width of the help windows will probably vary because installation programs on different platforms display a different number of characters on a line.

Parameter Definitions

The *Parameter Definition* section of the INS file specifies the configurable parameters for the driver and defines the valid options for each parameter. The parameter format is described in the next section.

If the network board can be configured by software and does not need to use the NET.CFG file, there should be no parameter definitions

Parameter Syntax

The NET.CFG file is created with parameter keywords and values. The INS file can define standard and/or custom parameters. The standard driver parameter keywords are: BUS ID, IRQ, PORT, MEM, DMA, SLOT, NODE ADDRESS, and FRAME (see *ODI Specification: DOS Client HSMs*, document version 4.0). The INS file can specify as many parameters as are needed to describe the available hardware and software options.

A parameter definition has the following format:

<ParameterCode>(<ParameterKeyword>)<ParameterName>
[<Parameter Help Text>]
<Valid Parameter Options>
$
$
$

Parameter Code

The *ParameterCode* field is eight characters long. The characters are defined as follows:

First character. The code's first character specifies the prompt format the installation utility will use to display the parameter on the screen for the user. This character will also affect how the <valid parameter options> are indicated. The possible values for the first character are listed below.

! = Select only one option from the list.
*** = Select more than one option from the list.**
$ = User input within a range (slightly different from above).
 # = Reserved for future use.

Second character. The code's second character indicates whether the user must select this parameter value or whether it is optional. This character also indicates whether a NET.CFG entry will be generated.

R (alpha) = Required. This parameter must be selected. A NET.CFG entry is always generated.
0 (numeric) = Optional. If nothing is selected, no NET.CFG entry will be generated.

Last characters. The last six characters indicate the bus type dependency of the parameter. The installation utility uses this code to determine whether the parameter is to be displayed (used), depending on the bus type of the machine. The *ParameterCode*'s 3 bus type characters are defined as follows:

3rd character I (alpha) = supports ISA adapters
0 (numeric) = does not support ISA adapters
4th character E (alpha) = supports EISA adapters
0 (numeric) = does not support EISA adapters
5th character M (alpha) = supports MCA adapters
0 (numeric) = does not support MCA adapters
6th character A (alpha) = supports PCMCIA adapters
0 (numeric) = does not support PCMCIA adapters
7th character P (alpha) = supports PCI adapters
0 (numeric) = does not support PCI adapters
8th character V (alpha) = supports VESA Local Bus adapters

0 (numeric) = does not support VESA Local Bus adapters

For example, a driver running on an ISA or EISA bus may require the PORT parameter. However, a driver running on an MCA machine may require only the SLOT parameter.

```
!RIE0000(PORT)Base I/O Port
$
$
$
!R00M000(SLOT)Slot Number
$
$
$
```

Parameter Keyword

The parameter's NET.CFG file keyword immediately follows the parameter code characters. The parameter keyword is enclosed in parenthesis and can be up to a maximum of 20 characters. The user-selected value for that parameter is placed in the NET.CFG file if no keyword (empty parenthesis) exists. These keywords can be custom keywords or any of the standard keywords listed in Table below.

Parameter Name

The parameter name follows the parameter keyword. This name is the title the installation utility displays to the user. The length of this field is limited to 30 characters. The parameter names are given in Table .

Parameter Help Text

A single line of optional help text may follow the parameter line. This line is preceded with a question mark (?) and can be a maximum of 256 characters long. You should only use help text with custom keywords, because the installation utility supplies help text for each standard keyword. Any help text must follow the keyword line and precede the parameter value lines described next in the NET.CFG file. If help is supplied with a standard keyword such as PORT, it will be appended to the default help text available from the installation utility.

The following table lists the standard parameter keywords and the default help text supplied by the installation utility:

Standard Parameter Keywords

Parameter Keywords	Parameter Names	Default Parameter Help Text
IRQ	**Hardware Interrupt**	**``Select the interrupt level that corresponds to the interrupt setting on the board or other device. The interrupt setting must be unique (one not used by another device in the machine)."**
PORT	**Base I/O Port**	**``Select the port value (base I/O address) that corresponds to the port address setting on the board or other device. Make sure the block of I/O addresses does not overlap the addresses of another device in the machine."**
MEM	**Memory I/O Address**	**``Select the memory address that corresponds to the memory setting on the board or other device. Make sure the block of memory addresses does not overlap the addresses of another device in the machine."**
DMA	**DMA**	**``Select the DMA channel that corresponds to the DMA setting on the board or other device. Make sure the DMA (Direct Memory Access) channel does not conflict with that of another device in the machine."**

SLOT	**Slot**	**``Select the slot number that corresponds to the expansion slot where the board or other device is installed."**
NODE ADDRESS	**Node Address**	**``Do not change this address unless you are prepared to administer local addresses according to the IEEE 802.2 specifications. The driver defaults to the node address on the board."**
FRAME	**Frame Type**	**``Select the frame type(s) used by the protocol(s) that your network requires. Make sure you have both the server and client configured for the same frame type."**
BUS ID	**Bus ID**	**``Select the bus type that corresponds to the bus used by the network interface card."**

Valid Parameter Options

Each line that follows the parameter description (up to the next parameter description and not including any help) is interpreted as a possible value for that parameter. Note that these are TEXT values and can be no longer than 20 characters. These values must also be in the same format as they would in the NET.CFG file.

The valid parameter values are specified in either *list* or *range* notation, depending on the first character of the *ParameterCode*. When you specify parameter values in a *list* form, you must enter each valid parameter value on a separate line.

Value *ranges* use only three value lines. The first value line is the beginning of the range. The second is the end of the range. The third is optional and is a default value preceded with an @ sign. If a default value is specified, the installation utility will generate the NET.CFG entry. If a default value is not specified, the installation utility will only generate a NET.CFG entry if the user enters a value. If the parameter is required, you must specify a third default value line.

The installation utility treats these values as strings. This allows the user to enter an L or M after the NODE ADDRESS parameter, designating in the NET.CFG file that the node address specified is in canonical (LSB) or noncanonical (MSB) format (see the *ODI Specification Supplement: Canonical and Noncanonical Addressing,* part number 107-000059-001).

Default Parameter Value

You designate a default parameter value in the INS file by preceding the value line with an `@' sign. In the example on the following page, the board setting defaults are INT 3 and PORT 300.

All required parameters must have a default specified. If a parameter is optional, you need not specify the default. The installation utility always displays a blank selection to the user for optional parameters. If no default is specified, then the "no value" option becomes the default. A NET.CFG entry is generated for any option other than the ``no value'' option. The installation utility does not generate a NET.CFG entry for a ``no value'' option.

The default settings in the driver INS file should match the default jumper settings of the network board as shipped by the manufacturer, where possible. If the jumper settings cannot be guaranteed, the defaults should match the most likely jumper settings, or the most common user settings.

Example INS File (NE2000)

```
InS_StArT
NE2000.COM
1.0
^Novell/Eagle NE2000,NE2000.COM,IE0000
?Please select the options that match your board's jumper settings.
!RIE0000(INT)Hardware Interrupt
2
@3
5
7
!RIE0000(PORT)Base I/O Port
@300
320
340
360
$OIE0000(NODE ADDRESS)Optional Node Address
0
FFFFFFFFFFF
```

```
*RIE0000(FRAME)Media Frame Type(s)
?
```

NOTE: Ethernet_802.2 is now the default frame type for ODI drivers. Existing LANs may be using Ethernet_802.3.

```
@Ethernet_802.2
Ethernet_802.3
Ethernet_II
Ethernet_SNAP
InS_EnD
```

Example INS File (using various options)

```
InS_StArT
PCN2.COM
1.0,ROUTE.COM
^IBM PC NetWork BroadBand or BaseBand Adapter
II,PCN2.COM,IE0000
^IBM PC NetWork BroadBand or BaseBand
Adapter/2,PCN2.COM,00M000,EFEF
?This board may be used with Source Routing. If your network is
using Source Routing simply load the ROUTE.COM file after the
PCN2.COM file has been loaded.
!RIE0000(PORT)Base I/O Port
@620
628
!O00M000(SLOT)Optional Slot Number
1
2
3
4
5
6
7
8
$OIEM000(NODE ADDRESS)Optional Node Address
0
FFFFFFFFFFF
!OIEM000()Primary or Alternate Adapter
?If your adapter is configured for Alternate, please select this option.
ALTERNATE
*RIEM000(FRAME)Media Frame Type
@IBM_PCN2_802.2 MSB
IBM_PCN2_802.2 LSB
IBM_PCN2_SNAP MSB
IBM_PCN2_SNAP LSB
InS_EnD
```

DRIVER.LST File

The DRIVER.LST file is an ASCII text file that lists the network boards the product supports. DRIVER.LST is located at the root of

the distribution diskette that contains the ODI driver files. It is specific to the drivers located on the distribution diskette. The installation utility uses DRIVER.LST to avoid the extensive overhead of scanning every .COM or .EXE file on the diskette for specific INS files. You do not need the DRIVER.LST file if the distribution diskette contains less than four .COM or .EXE files. (Note that this is all .COM or .EXE files and not just driver files.) If the installation program does not find a DRIVER.LST file, it will scan all .COM and .EXE files it finds on the distribution diskette to gather this information.

Each network board is listed on a separate line with four columns. The columns are delineated by a comma (,):

- **The first column is the full name of the network adapter and can contain a maximum of 48 characters.**
- **The second column is the filename (including the extension) of the driver used with the board and can contain a maximum of 13 characters.**
- **The third column is a six character code depicting bus type(s) supported for the adapter. The code is described below:**

  1st character I (alpha) = supports ISA adapters
  0 (numeric) = does not support ISA adapters
  2nd character E (alpha) = supports EISA adapters
  0 (numeric) = does not support EISA adapters
  3rd character M (alpha) = supports MCA adapters
  0 (numeric) = does not support MCA adapters
  4th character A (alpha) = supports PCMCIA adapters
  0 (numeric) = does not support PCMCIA adapters
  5th character P (alpha) = supports PCI adapters
  0 (numeric) = does not support PCI adapters
  6th character V (alpha) = supports VESA Local Bus adapters
  0 (numeric) = does not support VESA Local Bus adapters

  Note: Because all ISA boards work in EISA machines, `I' and `E' should both be used for ISA boards. This enables ISA boards installed in an EISA bus to appear in the installation utility. Keep in mind that ISA boards installed in EISA machines retain ISA functionality and features.

- **The fourth column is the *ProductID* field and applies only to MCA and EISA boards. This field contains the ID string stored in the POS registers in MicroChannel and EISA machines. The installation utility uses this string to automatically select the**

appropriate driver for the board found in the machine. This field is left blank for ISA boards.

Example DRIVER.LST File

```
Novell NE1000,NE1000.COM,IE0000
Novell NE2000,NE2000.COM,IE0000
Novell NE/2,NE2.COM,00M000,7154
Novell NE3200,NE3200.COM,IE0000,NVL0701
3Com EtherLink II,3C503.COM,IE0000
3Com EtherLink/MC,3C523.COM,00M000,6042
Novell RX-Net & RX-NET II,TRXNET.COM,IE0000
Novell RX-Net/2,TRXNET.COM,00M000,6014
IBM Token-Ring Network Adapter II & 16/4
Adapter,TOKEN.COM,IE0000
IBM Token-Ring Adapter/A,TOKEN.COM,00M000,E000
IBM Token-Ring 16/4 Adapter/A,TOKEN.COM,00M000,E001
IBM PC Network Baseband or Broadband Adapter
II,PCN2L.COM,IE0000
IBM PC Network Baseband or Broadband Adapter
II/A,PCN2L.COM,00M000,EFEF
Western Digital EtherCard PLUS Elite (all
cards),WDPLUS.COM,IE0000
```

Automatic
Hardware Detection:
Netware Plug and Play

Version 1.0 Changes for NetWare Disk and LAN Driver Specification -- Install Services Group --

(Updated for Rev 2.1d of the NWPA spec.)

draft

Introduction

The following changes are proposed to the Disk (NWPA) and LAN (CHSM) driver specifications, specifically for the purpose of automatic hardware detection. These changes will let drivers continue to work with previous versions of INSTALL.NLM, but will let future versions of INSTALL.NLM automatically match drivers to detected hardware. Drivers certified with, and after, the next major NetWare OS release will need to comply with these changes in order to be automatically selected.

draft

DDI and LDI Specification Changes for Autodetection

These changes affect the the *description files* that accompany the drivers (DDI/LDI files), rather than affecting the actual drivers. Important: A DDI or LDI file is *required* for driver certification by Novell Labs.

Card and Bus Identifiers

Currently (under the old specification), ISA, MCA, and EISA are the only bus types supported in server driver description files, and they are supported only in dependency expressions. Following is the proposed expanded list of bus identifiers:

- **ISA(standard ISA)**
- **PNPISA (ISA with a PnP BIOS, or a PnP Configuration Manager)**
- **EISA**
- **MCA**
- **PCI**
- **PCMCIA**

Product/Device Indentification (Prod:)

The PROD: label is part of the existing DDI/LDI specification. It enables an install/configuration utility to match a driver with a particular device. This new specification introduces *new* PROD: syntax that is compatible with the previous syntax and with the new bus types. Examples of old and new syntax are shown below.

Important: As of this specification release, the PROD: label is *required* for every bus type except ISA. However, if an ISA card has an EISA product ID, the PROD: label is *required*, and the EISA product ID should be used.

Old Syntax

Prod: <EISA Device ID>|<MCA POS ID>, ...

<EISA Device ID> = VVVPPPR,
<MCA POS ID> = HHHH

VVV = Vendor ID, uppercase letters A-Z
PPP = Product ID, hex characters
R = Revision number, hex character
HHHH = POS registers 1, 0

draft

New Syntax

PROD: '<card type>.<INCLUDE|EXCLUDE>.<id>', '...'

Enclose each product string in single quotes and separate multiple strings with commas. <INCLUDE|EXCLUDE> is optional for all types. If INCLUDE is specified and Install finds a match, the driver will be selected. If EXCLUDE is specified and Install finds a match, the module will not be selected. (If both INCLUDE and EXCLUDE are used – which is not generally recommended – EXCLUDE takes precedence. If neither INCLUDE nor EXCLUDE is used, INCLUDE is assumed.

- **PnP ISA and EISA syntax:**

  ' PNPISA.<INCLUDE|EXCLUDE>.VVV.PPP.R ' -- or --
  ' EISA.<INCLUDE|EXCLUDE>.VVV.PPP.R '

  VVV = Vendor ID, uppercase letters A-Z
  PPP = Product ID, hex characters
  R = Revision number, hex character

  Examples:
  PROD: ' PNPISA.INCLUDE.CPQ.055.1 '
  PROD: ' EISA.CPQ.067.0'

- **MCA syntax:**

  ' MCA.<INCLUDE|EXCLUDE>.HHHH '

  HHHH = Adapter ID (POS registers 1, 0), hex characters

  Example:
  PROD: ' MCA.8437 '
  PROD: ' MCA.EXCLUDE.8438 '

- **PCI syntax:**

  ' PCI.<INCLUDE|EXCLUDE>.VVVV.DDDD.NNNN.SSSS.RR '

  VVVV = Vendor ID, hex characters
  DDDD = Device ID, hex characters
  NNNN = Subsystem vendor ID, hex characters (0000 if pre-v2.1 PCI hardware)
  SSSS = Subsystem ID, hex characters (0000 if pre-v2.1 PCI hardware)
  RR = Revision number, hex characters

  Example:

PROD: ' PCI.8086.0202.0000.0000.00 '

- **PCMCIA syntax is yet to be determined, but will probably be:**

  ' PCMCIA.<INCLUDE|EXCLUDE>.VVVV.DDDD '

  The following data is from the card's CISTPL_MANFID tuple:

  vvvv = Manufacturer ID (TPLMID_MANF field), hex characters

  DDDD = Manufacturer Information (TPLMID_CARD field), hex characters

  Example:
  PROD: ' PCMCIA.1234.0032 '

Handling Special Characters in a PROD String
The DDI/LDI parser interprets an asterisk (*) or question mark (?) as a wildcard character, a period (.) as a field delimiter, and "\n" and "\t" as newline and tab characters. If you want to keep any of these interpreted characters as *literal* characters in the PROD string, you must precede (escape) each with a pound sign (#) so the parser will not strip or interpret them. Examples of handling these special characters are shown below.

Literal Char.	Example String
* or ?	Escape the character with a pound sign, such as: PROD: ' SCSI.00.venID.devID.V123#* '
\ (backslash)	Escape (precede) a "\n" or "\t" with another backslash, such as: PROD: ' SCSI.00.venID.devID.rev1\\NEW '
. (period)	Escape a period with a pound sign, such as: PROD: ' SCSI.00.venID.devID.V1#.23 '
# (pound sign)	Escape the pound sign with another pound sign, such as: PROD: ' SCSI.00.venID.devID.V##123 '

An asterisk (*) matches any and all characters from its position in the field to the end of a field. A question mark (?) matches a single character located at its position in the field. Epecially for EISA cards, if your driver can control different OEM revisions, you should use a wildcard to match the different revisions (for example, PROD: ' EISA.NVL.070.? '). Wildcarding is allowed for all fields except <INCLUDE/EXCLUDE>

Dependency Expressions: Handling Multiple Buses in One Machine

Unlike the previous specification, the new specification accommodates multiple buses in the machine for which the driver is intended. This means that some statements for a PCI/ISA machine will be interpreted differently with this new specification.

Consider the following expression:

```
PRINT if (BUS == ISA) OPT else HID { }
```

Under the old spec, the BUS expression evaluates to a *single* bus type. With the new spec, BUS may indicate a *set* of buses. The new specification makes the following interpretations:

- **In the expression (BUS == <bus type>), if <bus type> is any element of a BUS *set*, the expression evaluates to TRUE. Otherwise, it is FALSE.**
- **Similarly, in the expression (BUS != <bus type>), if <bus type> is not an element of a BUS set, the expression will evaluate to TRUE. Otherwise, it will be FALSE.**

Conclusions

1) **Because an EISA bus is also ISA-compatible, the expression (BUS==ISA) evaluates to TRUE on an EISA machine.**

2) **Because an ISA machine with PnP support is also ISA-compatible, the expression (BUS==ISA) evaluates to TRUE on an ISA machine with PnP support.**

3) **On an EISA machine with PnP support, the expressions (BUS==ISA), (BUS==PNPISA), and (BUS==EISA) will all evaluate to TRUE.**

Custom Device Driver Module Support

Custom device driver module support refers to an extension to the PROD: label that will be added to disk driver descriptions (DDI files) for NWPA CDMs. Used in this manner, the PROD: label indicates which modules to select and load as a result of a SCSI inquiry or an IDE identify command. The above description of escape sequences and wildcards applies to SCSI and IDE PROD: string fields as well.

Important: For NWPA CDMs, the PROD: label is *required* in the DDI file.

Example
The following assumes an adapter driver (HAM) has been loaded. Install will load the

HAM and get information from NWPA to create a list of device types associated with the adapter. Next, Install examines all available DDI files for CDMs and tries to match the device type and module. It then loads the module(s) that best match the devices found.

For example, assume a tape device was found after doing a SCSI inquiry. Following is a driver description for TAPEDAI. The inquiry command will find a match in the TAPEDAI.DDI file and then load TAPEDAI.DSK, as illustrated below.

```
Driver TAPEDAI
{
    ...description and other labels...

    PROD: ' SCSI.INCLUDE.01. *. *. * '
    ...parameters...
}
```

The PROD: label with the SCSI designation indicates the device types supported by the TAPEDAI driver.

SCSI Extension Syntax

The complete syntax for the SCSI extension of the PROD: label is shown below. (The PROD: label may exist multiple times for different combinations of type numbers, device product IDs, etc.)

```
PROD:  ' SCSI.<INCLUDE|EXCLUDE>.
    <device type number>.<device vendor ID>.
     <device product ID>.<device product revision> ', ' ... '
```

Field	Description
INCLUDE	If Install finds a match, the driver will be selected.
EXCLUDE	If Install finds a match, the module will not be selected. If neither INCLUDE nor EXCLUDE is specified, INCLUDE is assumed. (If both INCLUDE and EXCLUDE are used – which is not generally recommended – EXCLUDE takes precedence.)
<device type number>	Device or hardware type listed below; should correspond to the Peripheral Device Type obtained from a SCSI Inquiry.

00 Direct-access device (e.g., magnetic disk)
01 Sequential-access device (e.g., magnetic tape)
02 Printer device
03 Processor device

04 Write-once device (e.g., some optical disks)
05 CD-ROM device
06 Scanner device
07 Optical memory device (e.g., some optical disks)
08 Medium Changer device (e.g., jukeboxes)
09 Communications device
*** Wildcard, matches any device type**

<vendor ID> **Vendor identification string; 1-8 bytes, ASCII. An asterisk (*) matches any vendor ID. <vendor ID> should correspond to the Vendor Identification obtained from a SCSI Inquiry.**

<product ID> **Product identification string; 1-16 bytes, ASCII. An asterisk (*) matches any product identifier. <product ID> should correspond to the Product Identification obtained from a SCSI Inquiry.**

<product revision> **Product revision string; 1-4 bytes, ASCII. An asterisk (*) matches any product revision. <product revision> should correspond to the Product Revision Level obtained from a SCSI Inquiry.**

IDE Extension Syntax

The complete syntax for the IDE extension of the PROD: label is shown below. (The PROD: label may exist multiple times for different combinations of type numbers, model numbers, and firmware revision levels.)

```
PROD: ' IDE.<INCLUDE|EXCLUDE>.
    <device type number>.<model number>.<firmware revision>', ' ... '
```

Field	Description
INCLUDE	If Install finds a match, the driver will be selected.
EXCLUDE	If Install finds a match, the module will not be selected. (If both INCLUDE and EXCLUDE are used – which is not generally recommended – EXCLUDE takes precedence. If neither INCLUDE or EXCLUDE is used, INCLUDE is assumed.).
<device type number>	Device or hardware type listed below; should correspond to the Peripheral Device Type obtained from a SCSI Inquiry.
00	Direct-access device (e.g., magnetic disk)

01	**Sequential-access device (e.g., magnetic tape)**
02	**Printer device**
03	**Processor device**
04	**Write-once device (e.g., some optical disks)**
05	**CD-ROM device**
06	**Scanner device**
07	**Optical memory device (e.g., some optical disks)**
08	**Medium Changer device (e.g., jukeboxes)**
09	**Communications device**
*****	**Wildcard, matches any device type**

<model number>	**Model number string; 1-40 bytes, ASCII. An asterisk (*) matches any model number identifier. <model number> should correspond to the Model Number obtained from an IDE Identify.**
<firmware revision>	**Firmware revision string; 1-8 bytes, ASCII. An asterisk (*) matches any firmware revision. <firmware revision> should correspond to the Firmware Revision obtained from an IDE Identify.**

Examining the Driver Description Files
After all adapter drivers have been selected and copied, Install will load the adapter drivers. For each adapter driver, Install will then do a SCSI inquiry or an IDE identify.

For each device found during the SCSI inquiry or IDE identify, Install will search all available driver description files for all the PROD: labels. If the driver matches an EXCLUDE item, that driver description will be skipped. Alternatively, if the driver matches an INCLUDE item, it will be given a fit score.

There may be multiple matches of a device in different driver descriptions. These will be resolved as follows: exact matches get a higher fit score than wildcards, and exact field matches of the first fields (in the PROD: declaration) get a higher score than later ones. The driver description with the highest score will be selected and loaded (if not already loaded).

Guidelines for Composing Driver Help Information

Because the INSTALL.NLM user interface may change from time to time, adding INSTALL.NLM screen navigation information in the driver help information of DDI/LDI files can cause problems. A new guidelines section will be added to the DDI/LDI specification sections in both the NWPA and CHSM specifications. This will

clarify what is expected in help information, as well as give other suggestions for writing high-quality DDI/LDI files.

draft

General Driver Specification/Conventions

The following specifications changes improve usability of automatic detection and user interaction in general.

Monolithic Disk Drivers In The Next Release

Monolithic drivers, such as DDFS drivers, will be supported in the next release ofNetWare. Automatic hardware detection will work with these drivers if the appropriate changes are made in the driver and its associated DDI. See the RDFIRST.WPD file associated with the NWPA Functional Specification and Developer's Guide for details on how this is to be done.

As already stated, the user should only have to select one driver for hard disk support. DDFS drivers should discover devices and load subsequent support modules.

It is preferable that the DDFS drivers also automatically attempt to load support for modules other than disk (such as tape).

Driver Load Instances

There should be one physical load instance per *physical* adapter or card, unless the driver has no command-line parameters. In that case, a single load instance is sufficient. (Multifunction LAN cards already require one load instance per adapter or per function.) This lets the user know what load line corresponds to which card so the desired card can be disabled. There may be multiple *logical* load instances per physical card.

draft