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.''

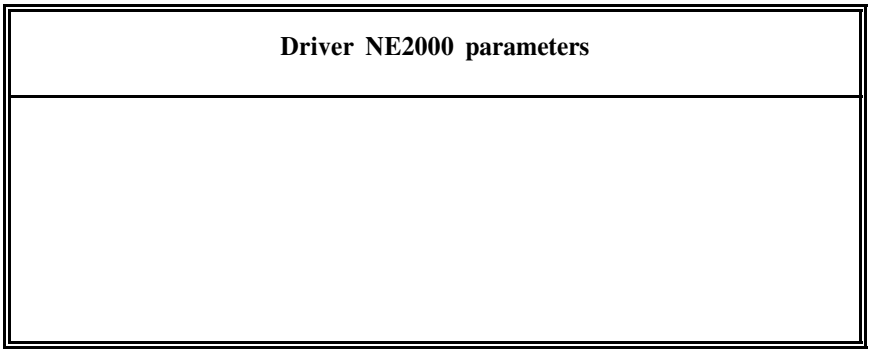
The example below shows a sample PROMPT parameter block using the local predefined parameter, INT, and the resulting screen displayed in the installation utility.

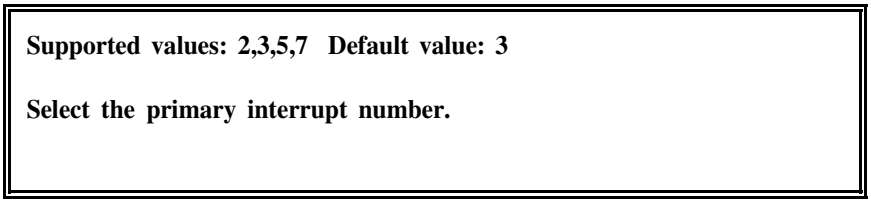
Note: **The *Description, Help,* and *Type* fields shown below are included to illustrate their use in the PROMPT example. For the local predefined INT parameter, these fields have default values and are not usually required.**

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:





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	**PROMPT**	**Primary interrupt**
INT2	**PROMPT**	**Second interrupt**
PORT or PORT1	**PROMPT**	**Primary I/O port**
PORT2	**PROMPT**	**Second I/O port**
MEM or MEM1	**PROMPT**	**Primary memory address**
MEM2	**PROMPT**	**Second memory address**
DMA or DMA1	**PROMPT**	**Primary DMA address**
DMA2	**PROMPT**	**Second DMA address**
SLOT	**PROMPT**	**Machine slot number**
NODE	**PROMPT**	**Node address**
RETRIES	**PROMPT**	**Number of retries**
CHANNEL	**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