REFERENCE MANUAL

# NaturalLink™ Toolkit

Part No. 2240316-0001 *C
August 1985

TEXAS INSTRUMENTS

# MANUAL REVISION HISTORY

NaturalLink™ Toolkit Reference Manual (2240316-0001)

| | |
|---|---|
| Original Issue | December 1983 |
| Revision | April 1984 |
| Revision | November 1984 |
| Revision | August 1985 |

The computers, as well as the programs that TI has created to use with them, are tools that can help people better manage the information used in their business; but tools—including TI computers—cannot replace sound judgment nor make the manager's business decisions.

Consequently, TI cannot warrant that its systems are suitable for any specific customer application. The manager must rely on judgment of what is best for his or her business.

The system-defined windows shown in this manual are examples of the software as this manual goes into production. Later changes in the software may cause the windows on your system to be different from those in the manual.

Printed in U.S.A.

# ABOUT THIS MANUAL

This manual describes the NaturalLink™ Toolkit, a set of utilities that simplifies and speeds the development of NaturalLink natural language interfaces. The manual presents a broad overview of NaturalLink user software and provides a detailed technical discussion of the Toolkit components used to produce software packages. This manual is intended for the software application programmer/designer.

This manual contains the following chapters and appendices.

**Chapter 1:** Introduction — Contains an overview of NaturalLink technology, NaturalLink Toolkit utilities, and user software.

**Chapter 2:** Using the NaturalLink Toolkit — Outlines the procedures for creating a typical NaturalLink interface.

**Chapter 3:** Creating NaturalLink Sentences — Describes how to develop sentences for a NaturalLink application.

**Chapter 4:** NaturalLink Grammar File — Describes a formal context-free grammar and the NaturalLink grammar format, and explains how NaturalLink software uses the grammar file.

**Chapter 5:** Creating an NLmenu Screen — Describes how to create the main NaturalLink menu (the NLmenu screen from which the user builds the sentences to be parsed and translated into the target language or command).

**Chapter 6:** Using the NaturalLink Interface Builder — Provides an overview of the NaturalLink tools for building and testing interfaces.

**Chapter 7:** Testing the NaturalLink Grammar — Describes the five syntax-checking tests performed on the grammar by the interface testing tools.

**Chapter 8:** Generating the NaturalLink Lexicon — Describes the interactive lexicon-building procedure and discusses each option in the Generate or Modify Lexicon menu.

**Chapter 9:** Application Control of User Options — Describes two methods of controlling user options: user-defined experts and dynamic lexical items. Discusses the types of problem each method is designed to solve and provides step-by-step procedures for each.

**Chapter 10:** Generating NaturalLink Sentences — Describes the Generate Representative Sentences option and explains the format of the resulting generated sentences output file.

**Chapter 11:** Checking an Interface Screen Against Its Lexicon — Describes the screen/lexicon consistency check utility. This utility looks for incompatibilities between the screen and lexicon files that would prevent the interface file from being successfully generated.

**Chapter 12:** Specifying Help and Window Coordinates for Sessioner Windows — Describes the process of attaching Help messages and specifying window coordinates for windows used by the Sessioner.

**Chapter 13:** Generating the Interface File and Testing Translations — Explains how to generate the interface file and test the translations returned for a set or subset of representative NaturalLink sentences.

**Chapter 14:** Other Interface Builder Options — Describes the three Interface Builder options that display files and screens.

**Chapter 15:** The NaturalLink Sessioner — Discusses the operation of the NaturalLink Sessioner (driver), describes the Sessioner's command options, and explains how to use the Toolkit's high-level language interface routines to execute the Sessioner.

**Chapter 16:** Interface Customization Feature — Discusses how the user can modify the interface by editing phrases and adding synonyms to windows. Presents information for the user and application designer on the use of this feature.

**Chapter 17:** Final Steps in Preparing the NaturalLink Interface — Discusses how to package the application after the interface file is built and all code is linked, and lists the files to back up.

**Appendix A:** Equipment Requirements — Describes the NaturalLink Toolkit hardware and software environment.

**Appendix B:** NaturalLink Error Codes — Lists and explains the NaturalLink error messages.

**Appendix C:** High-Level Language Interface — Contains information on calling and linking the NaturalLink software with the Lattice C™, MS®-FORTRAN, MS-Pascal, and MS-BASIC Compilers.

**Appendix D:** NaturalLink Demonstration Program — Contains a Lattice C program for tailoring a demonstration of the NaturalLink interface and Window Manager.

Lattice C Compiler is a trademark of Lattice Incorporated.

MS is a registered trademark of Microsoft Corporation.

The following documents contain information relevant to the Texas Instruments Professional Computer and the NaturalLink Toolkit.

| Title | Part Number |
|---|---|
| *MS-DOS Operating System, Volume I* | 2243310-0001 |
| *MS-DOS Operating System, Volume II* | 2243311-0001 |
| *NaturalLink Window Manager Reference Manual* | 2240317-0001 |
| *NaturalLink Technology Workbook* | 2243736-0001 |

# CONTENTS

# INTRODUCTION 1

## NaturalLink Technology

**1.1** Texas Instruments NaturalLink natural language technology is designed to increase the productivity of computer users who have little or no training or experience in programming. NaturalLink technology accomplishes this by simplifying the interface between users and application programs. With the software tools provided in the NaturalLink Toolkit, you, the application designer, can produce windows, menus, and menu items that enable users to interact with the computer by selecting uncomplicated options. These options consist of familiar words or phrases —the building blocks of all natural languages—that the user can assemble to build command sentences or to issue instructions directly to an application program.

Because the NaturalLink software is data-driven, you can tailor selection options to the language and task requirements of specific users, whether they are accountants, corporate vice-presidents, industrial plant workers, lawyers, or secretaries. NaturalLink software can perform a variety of computer-related tasks, such as database query, industrial systems control, and electronic mailing.

An important feature of the NaturalLink software is that it controls the order of words and phrases that a user selects from a menu. The user can formulate only queries or sentences that produce valid commands for the underlying system or application. System stability is enhanced because applications cannot receive invalid directives. Time-consuming error checks are reduced, and response time is improved.

**NOTE:** For reasons of clarity and consistency, English is the language referred to throughout this manual in discussions of natural language interfaces. It should be emphasized that the NaturalLink Toolkit can be used to develop interfaces for any natural language; NaturalLink interfaces have, in fact, been developed in French, German, Italian, and other languages.

## Windows and Menus

**1.2** The NaturalLink software enables you to create a variety of windows and menus. The simplest of these might consist of a single window containing options the user can select to perform a computer-related task.

More complex menus, with multiple windows and various arrangements of window items and labels, can be constructed to meet application requirements. These menus might direct the user to type specific information or to construct command sentences with nouns, verbs, objects, or other sentence elements selected from the interactive menu lists.

The NaturalLink software controls each step of menu selection and sentence construction. The user makes menu selections from active windows. To create a command sentence, the user places the cursor on the appropriate word or phrase in each active window and presses the **ENTER** key. The program then examines the sentence under construction and decides which window(s) should be activated next and which choices should be displayed. Because the choices are controlled in this way, the user has no opportunity to make invalid selections. Thus, grammar, punctuation, lexical, and typing errors are eliminated. The software translates the error-free input into the application command language and sends it to the application.

## Toolkit Components

**1.3** Texas Instruments NaturalLink Toolkit is a set of interactive software tools that helps you create the data files necessary to produce high-level interfaces for almost any application. The NaturalLink Toolkit is packaged on nine diskettes as follows:

- Window Manager Utilities diskette A — Contains two utilities:
  - Screen Builder (SBUILD) — An interactive utility with which you can specify the appearance and behavior of a NaturalLink screen.
  - Message Builder (MBUILD) — An interactive utility that helps you construct an error/Help message file for use with the NaturalLink Message Manager.
- Window Manager Utilities diskette B — Contains two utilities:
  - Function Key Change Utility (KBUILD) — An interactive utility with which you can change the default function keys used by the Window Manager and NaturalLink Sessioner.
  - Phrase Editing Utility (PBUILD) — An interactive utility with which you can change all the default phrases the NaturalLink run time uses to display information and options to the user.

- Window Manager Runtime Object diskette — Contains the object code for a set of high-level language interface routines and the NaturalLink object for the Window Manager.

- Window Manager Demonstration diskette — Contains the Window Manager demonstration program, which illustrates several Window Manager features and function calls. This diskette provides a demonstration that includes source, object code, and link streams for each of the high-level languages supported by NaturalLink software.

- Toolkit Utilities diskette A — Contains the menu-driven Interface Builder (IBUILD) utility with which you can test the interface grammar for format errors and syntactic and semantic well-formedness, generate and modify a lexicon, generate the interface file, and test the generation of target language translations.

- Toolkit Utilities diskette B — Contains support files (screen and message files) for the IBUILD utility.

- Toolkit Runtime Object diskette — Contains the object code for the NaturalLink Sessioner. The object code on this diskette must be used in conjunction with the Window Manager run-time object code.

- Toolkit Demonstration diskette A — Contains a NaturalLink driver demonstration program that provides a means of testing any interface you have created and illustrates the use of the NaturalLink Sessioner (driver). The diskette contains the source, link streams, and a linked .EXE file for this demonstration.

- Toolkit Demonstration diskette B — Contains a NaturalLink demonstration program that is a complete working interface to a database. This demonstration incorporates most of the NaturalLink features provided by the Toolkit. The source code of the demonstration program is provided on diskette A so you can see how these Toolkit features are actually coded and used.

## Creating the Interface File

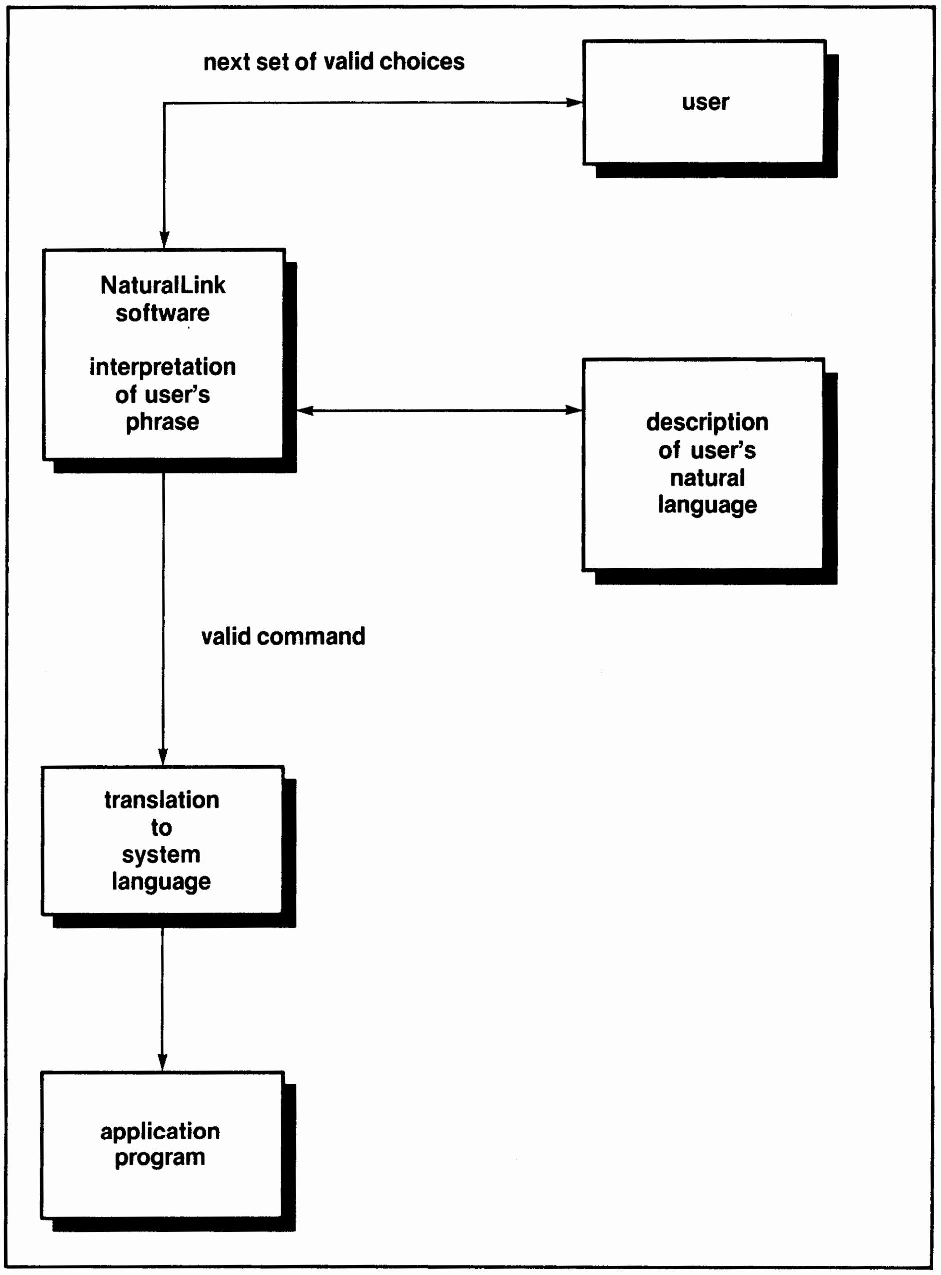
**1.4** The interface file (a binary file) drives the NaturalLink software and governs user input to the application. The interface file indicates to the NaturalLink software which windows to activate and what data to display in each window.

You can generate the interface file only after creating and testing the following files:

- Grammar file — Contains rules that specify how words and phrases in the natural language can be combined, in what order they can appear, and how they should be mapped to the target language of the application program.
- Lexicon file — Contains the elements from the NaturalLink grammar file, the natural language words and phrases the user will select, the labels of the windows in which those words and phrases will appear, and the target language translations that will be sent to the application program.
- Screen description file — Contains the set of data structures that tell the NaturalLink Window Manager how to design and place the windows.

Toolkit components help you develop the interface file by providing an automated method to test and debug the grammar, produce a lexicon, and create and control the screen description. Figure 1-1 illustrates how NaturalLink software, linked with an application program, first accepts input from a user, then translates that input into a target string, and, finally, sends it to the application program.

**Figure 1-1** **NaturalLink Interface**


next set of valid choices
user
NaturalLink software
interpretation of user's phrase
description of user's natural language
valid command
translation to system language
application program

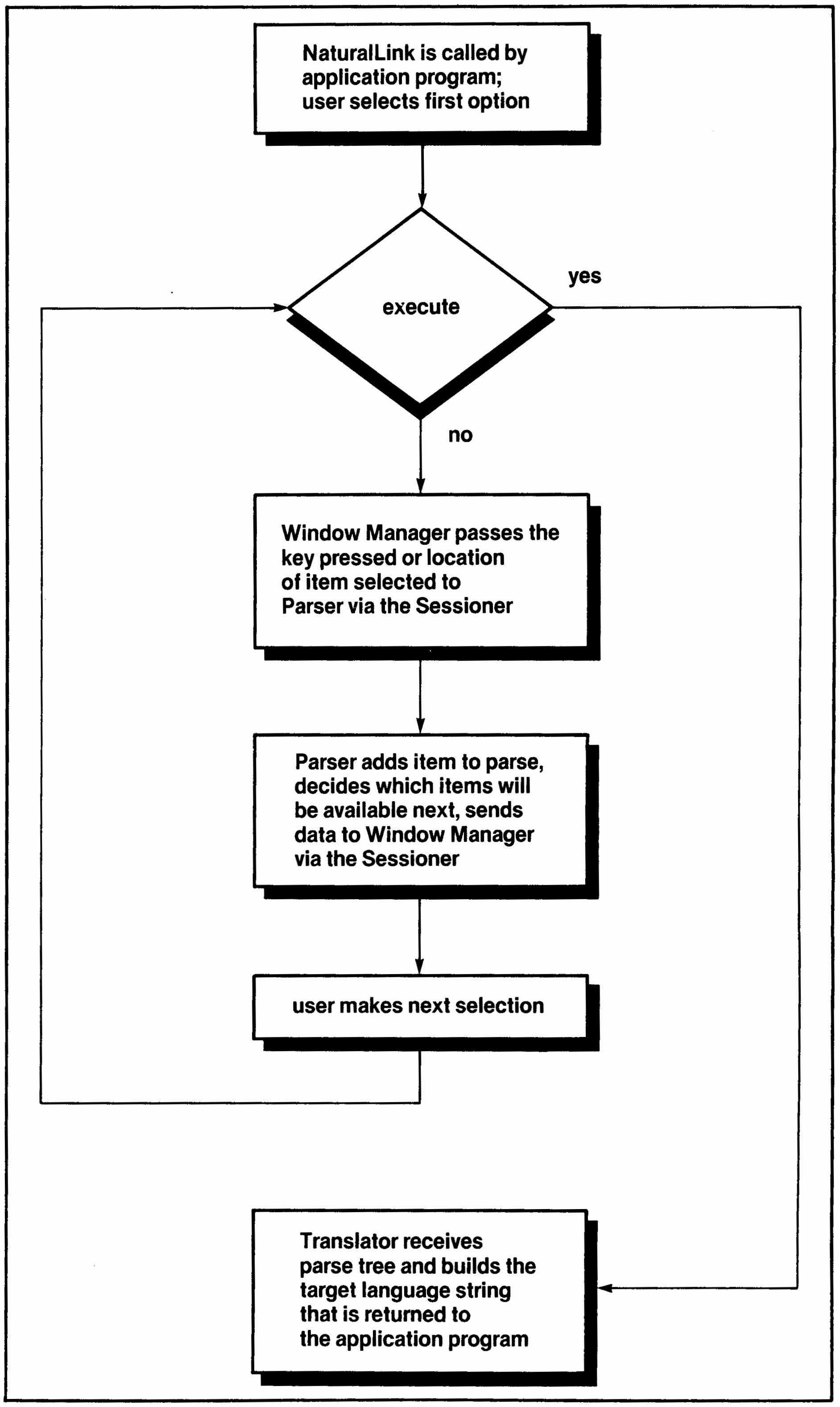
## User Interaction

**1.5** During execution of an application program that uses NaturalLink software, four components handle the interaction between the user and an application. They are:

- Window Manager — Controls the appearance of the screen, the movement of the cursor, and the selection of items within windows.
- Parser — Keeps track of user choices and decides the next logical choices.
- Translator — Constructs the target language string from the parsed input sentence.
- Sessioner (driver) — Controls the interaction among the Window Manager, the Parser, the Translator, and the application program.

Figure 1-2 illustrates how these components interact during the execution of an application.

**Figure 1-2** **Typical NaturalLink Flow of Control**


NaturalLink is called by application program; user selects first option
execute
yes
no
Window Manager passes the key pressed or location of item selected to Parser via the Sessioner
Parser adds item to parse, decides which items will be available next, sends data to Window Manager via the Sessioner
user makes next selection
Translator receives parse tree and builds the target language string that is returned to the application program

# USING THE NATURALLINK TOOLKIT 2

## Introduction

**2.1** A series of steps and procedures must be followed to produce each component in a complete NaturalLink interface. NaturalLink software helps check the validity and compatibility of each interface component as you develop it. Most components are interdependent and serve as input for generating subsequent components. You may sometimes need to return to a previous step in the development process, because the modification of one component often affects a previously generated component.

## Interface Development

**2.2** A typical interface development procedure is as follows:

1. Copy all NaturalLink Toolkit diskettes to the Winchester disk. Assign the Winchester drive as the default drive.

2. Create a set of sentences. Study the application, and write sentences that best represent the capabilities of the application. Refer to Chapter 3, Creating NaturalLink Sentences, for guidelines.

3. Develop and test the screens. With the Screen Builder utility, develop and test the window(s) and screen(s) needed for your interface. For detailed information about the Screen Builder utility, refer to the *NaturalLink Window Manager Reference Manual.* Chapter 5, Creating an NLmenu Screen, of this manual contains information about Screen Builder.

4. Write a grammar that defines the set of sentences created in step 2. Refer to the *NaturalLink Technology Workbook* for detailed instructions on writing a grammar for the NaturalLink software.

5. Develop grammar translation information. Using the formats in Chapter 4, The NaturalLink Grammar File, and the procedures outlined in the *NaturalLink Technology Workbook*, develop the translation information for the grammar created in step 4.

6. Enter the grammar, along with the translation information, into your computer. Using a standard text editor, enter your grammar into a sequential file on your disk.

7. Specify the new working interface. Invoke the Interface Builder utility, and use the Record a New Interface's Information option to define a working interface for the grammar. Refer to Chapter 6, Using the NaturalLink Interface Builder, for details.

8. Test the grammar. Use the Test the Grammar for the Interface option in the Interface Builder utility to test the grammar. Refer to Chapter 7, Testing the NaturalLink Grammar, for information about this procedure.

9. Build the lexicon. Use the Generate or Modify the Lexicon for the Interface option of the Interface Builder utility to construct the lexicon. Refer to Chapter 8, Generating the NaturalLink Lexicon, for details.

10. Generate sentences. Use the Generate Representative Sentences for the Interface option to produce a set of sentences you can analyze to discover errors not found during the grammar testing procedure.

11. Check screen and lexicon consistency. Use the Check the Screen Against the Lexicon option of the Interface Builder utility to ensure that all windows referenced by the lexicon are in the NLmenu screen file and contain the correct attributes. Refer to Chapter 11, Checking an Interface Screen Against Its Lexicon, for details.

12. Specify Help and window coordinates for Sessioner windows. If you want to attach Help messages or specify certain window coordinates for the windows used by the Sessioner, you should do so before generating the interface file. Refer to Chapter 12, Specifying Help and Window Coordinates for Sessioner Windows, for details.

13. Generate the interface and test your translations. Use the Generate the Interface File option of the Interface Builder utility to build your interface. Then, using the Test the Interface's Translations option, test the translations for your generated sentences. Refer to Chapter 13, Generating the Interface File and Testing Translations, for details.

14. Link the application. After you have tested the interface, modify your application to make the necessary calls to the NaturalLink software. Refer to Chapter 15, The NaturalLink Sessioner, and Appendix C, High-Level Language Interface, for details.

15. Build the user diskette and save your files. After you have linked the application with the NaturalLink software, build the user diskette and save the grammar, lexicon, expert, screen, and other files in case you enhance your interface in the future. Refer to Chapter 17, Final Steps in Preparing the NaturalLink Interface, for details.

The structure of this manual reflects the typical interface development procedure just described. Information is presented in the order you will need it as you create your NaturalLink interface. The NaturalLink Toolkit can produce interfaces for a variety of computer applications, but for consistency this manual presents an extended example of step-by-step procedures for developing an interface to a relational database.

# CREATING NATURALLINK SENTENCES 3

## Introduction

**3.1** The first step in the interface development process is to study the computer application carefully and then write a set of clear, natural language sentences that reflect all the capabilities of the target language for that application. In this case, the data manipulation language (DML) for a relational database is the target language. This chapter describes some factors you should consider and some problems you may encounter in developing a set of natural language sentences for a particular data manipulation language.

## Creating Natural Language Sentences

**3.2** Assume your database consists of various tables that the user can access by typing in the proper DML commands. Tables can be linked, and multiple tables can be accessed by a single command phrase. One table contains information about cars, another contains demographic information, and so forth.

Following are examples of DML commands and natural language sentences. The numbers in parentheses identify examples. The first two digits refer to the paragraph in which the examples occur, and the next digit is the example number within the paragraph.

In the first example, assume you have the DML commands LIST and COUNT, and you wish to create natural language sentences that will enable a user to access information about cars:

| | | | |
|---|---|---|---|
| (3.2.1)* | (1) | LIST | field_name |
| | (2) | COUNT | field_name |

Here, LIST means to display values for the specified field, and COUNT means to count the number of times a field value exists. In this case, LIST CARS will show all the cars in the database; COUNT CARS will show the number of cars in the database. The following sets of natural language sentences can be created from this simple DML:

(3.2.2) (1) List all cars.
(2) Count the number of cars.

(3.2.3) (1) Display all cars.
(2) Display the number of cars.

(3.2.4) (1) Show all cars.
(2) Show the number of cars.

(3.2.5) (1) Find all cars.
(2) Find the number of cars.

* The numbers in parentheses identify examples. The first two digits refer to the paragraph in which the examples occur, and the next digit is the example number within the paragraph.

An important factor to remember is that the simplest sentences are generally the best sentences. Not only are simple sentences easier for a user to understand, but you will find that writing grammar rules for them is easier. Although all the preceding sentences are easy to understand, the sentences in 3.2.2 require the use of two different verbs, while the sentences in 3.2.3 through 3.2.5 use a single verb for both LIST and COUNT. Thus, the last three sets of sentences are recommended because they require fewer grammar rules.

Now, look at the following DML commands:

```
(3.2.6)    (1) LIST       field_name_1
               BY         field_name_2

           (2) LIST       field_name_1
               BY HIGHEST    field_name_2
```

Here, the first DML command (3.2.6.1) means to display the value of field_name_1 in ascending order of the value of field_name_2, while the second command (3.2.6.2) means to display the value of field_name_1 in descending order of the value of field_name_2. With STATES for field_name_1 and POPULATION for field_name_2, the following natural language sentences can be written:

(3.2.7) (1) Show states in order of population,
    showing the least populous states first.
(2) Show states in order of population,
    showing the most populous states first.

(3.2.8) (1) Show states sorted by population
    in ascending order.
(2) Show states sorted by population
    in descending order.

(3.2.9) (1) Show states in ascending order of population.
(2) Show states in descending order of population.

Although the sentences in 3.2.7 sound natural, it would be difficult to write a grammar to generate such complicated sentences. The sentences in 3.2.8 cannot be recommended either, because it is not clear whether the phrases "in ascending order" and "in descending order" refer to "states" or to "population." Ambiguous sentences are best avoided, because they are confusing to the user and because they require special grammar rules to resolve their meanings.

The sentences in 3.2.8 have another drawback. Because the field_name ("population") splits the sort-phrase into two parts (that is, "sorted by" field_name "in ascending order"), two separate windows may be necessary, one to display each phrase. Screen space is limited and should not be filled unnecessarily. The sentences in 3.2.9, on the other hand, are both clear and concise, and they do not require unusual window configurations. They are the best sentences for the purpose.

## Writing Unambiguous Sentences

**3.3** To avoid ambiguities, it is sometimes necessary to repeat two or more elements in a sentence. Consider the following sentence:

(3.3.1) Find jobcards that list operations whose operation number is equal to 50 and whose order number is greater than 100.

Assume that both the JOBCARD table and the OPERATION table have a field called order_number. In sentence 3.3.1, the phrase "whose order number is greater than 100" could refer to either "jobcard order number" or "operation order number." The sentences in 3.3.2 clear up this ambiguity.

(3.3.2) (1) Find jobcards that list operations whose operation number is equal to 50 and whose jobcard order number is greater than 100.

(2) Find jobcards that list operations whose operation number is equal to 50 and whose operation order number is greater than 100.

Following are further examples of unambiguous sentences:

(3.3.3) (1) Find workers who have jobcards whose jobcard order number is 20 and whose worker employee number is 150.

(2) Find workers who have jobcards whose jobcard order number is 20 and whose jobcard employee number is 150.

(3.3.4) (1) Find operations that are listed on jobcards whose jobcard hours are 10 and whose operation piece number is 200.

(2) Find operations that are listed on jobcards whose jobcard hours are 10 and whose jobcard piece number is 200.

(3.3.5) (1) Find workers whose worker name is Steve Jones.
(2) Find jobcards whose jobcard piece number is 300.

## NaturalLink Experts

**3.4** Some of the sentences in the preceding examples require specific numeric values. The user must be able to fill in the values. To this end, the NaturalLink software contains several *expert* fields in which the user can enter values. For further information on the various expert routines available in the NaturalLink software, refer to Chapter 8, Generating the NaturalLink Lexicon, and to Chapter 9, Application Control of User Options.

# THE NATURALLINK GRAMMAR FILE 4

| *Paragraph* | *Title* | *Page* |
|---|---|---|

## Introduction

**4.1** Language is governed by rules. In the NaturalLink grammar file, the rules of a natural language are expressed in a formal notation that a computer can use to generate sentences for a NaturalLink interface. This chapter discusses the basic rules and conventions for creating a NaturalLink grammar file. For a more detailed explanation, refer to the *NaturalLink Technology Workbook*.

The general procedure for writing a NaturalLink grammar consists of two steps:

1. Write a context-free grammar defining the natural language.

2. For each rule, construct translation lists that tie together the grammar, the lexicon, and the target string.

The rest of the chapter explains these steps and discusses how NaturalLink uses the grammar file.

## Context-Free Grammar

**4.2** Linguists have developed symbolic conventions for describing the structure of language. Here is a representation of a context-free grammar, that is, a grammar of the following form:

| | | | | |
|---|---|---|---|---|
| Grammar (1) | (1.1) | S | $\rightarrow$ | NOUN VERB |
| | (1.2) | NOUN | $\rightarrow$ | birds |
| | (1.3) | VERB | $\rightarrow$ | fly |

Each rule in a context-free grammar has a single element on the left side connected by an arrow to one or more elements on the right side. The element 'S' stands for "sentence." The arrow is interpreted as meaning "may consist of." The preceding grammar rules are read as follows:

> A sentence 'S' may consist of 'NOUN' followed by 'VERB'; 'NOUN' may consist of 'birds'; 'VERB' may consist of 'fly'.

Each element in the rule is called a rule element. 'S', 'NOUN', 'VERB', 'birds', and 'fly' are all rule elements. Elements that can appear on both the left and the right sides of the rules are called nonterminal elements. 'S', 'NOUN', and 'VERB' are nonterminal elements. Elements that can appear *only* on the right side of the grammar rules are called terminal elements. The elements 'birds' and 'fly' are terminal elements, since they do not appear on the left side of any rule. (Terminal elements are also called lexical items or lexical elements because they are further defined in the lexicon, the language's list of words and their definitions.) Throughout this chapter, capital letters represent nonterminal elements and lowercase letters represent terminal elements.

## How Grammar Rules Control Sentence Generation

**4.2.1** Grammar rules prescribe the structure of the sentences that can be generated. Assume you have the following set of grammar rules for a language:

| | | |
|---|---|---|
| Grammar (2) | (2.1) | $S \rightarrow A\ B\ C$ |
| | (2.2) | $A \rightarrow a\ C$ |
| | (2.3) | $B \rightarrow b$ |
| | (2.4) | $C \rightarrow c$ |

Following is an informal description of these grammar rules:

> A sentence 'S' consists of nonterminal elements 'A', 'B', and 'C'. 'A' consists of terminal element 'a' and nonterminal element 'C'. 'B' consists of terminal element 'b'. 'C' consists of terminal element 'c'.

The following example illustrates a systematic method of applying these grammar rules to generate a sentence:

```
S
A   B C     if you apply rule (2.1)
a C B C     if you apply rule (2.2)
a C b C     if you apply rule (2.3)
a c b c     if you apply rule (2.4)
```

First, write the element 'S'. Directly below that write the sequence of elements that appear on the right side of the S rule (2.1). If the grammar had more than one rule with S on the left side, you could have picked any S rule. On the third line, expand one of the nonterminal elements in the second line. In the preceding example, the third line, 'a C B C', was derived from the second line by replacing the element 'A' with the sequence of elements 'a C' that appears on the right side of the A rule (2.2). Repeat this procedure until there are no more rules to apply, that is, until all the nonterminal elements have been expanded. The resulting derivation is in *table* form.

When all nonterminal elements in a sentence have been expanded, it can be said that a sentence has been generated by the grammar. Frequently, more than one element in a given line can be rewritten to create the next line. In the preceding example, three elements on the second line, 'A', 'B', and 'C', can be expanded. The order in which you expand the elements does not alter the ultimate outcome of the sentence, as you can see by comparing the preceding example with the following one.

| | |
|---|---|
| S | |
| A    B C | if you apply rule (2.1) |
| A    b C | if you apply rule (2.3) |
| A    b c | if you apply rule (2.4) |
| a C b c | if you apply rule (2.2) |
| a c b c | if you apply rule (2.4) |

Note that rule 2.4 had to be applied twice in order to reduce all elements to terminal elements.

Whenever more than one rule in the grammar can be applied to a particular element, the outcome of the expansion depends upon which rule is chosen. For example, if you add a rule (2.5) to the grammar, the grammar can generate either of the sentences shown in the next example.

| | | |
|---|---|---|
| Grammar (2)' | (2.1) | S $\rightarrow$ A B C |
| | (2.2) | A $\rightarrow$ a C |
| | (2.3) | B $\rightarrow$ b |
| | (2.4) | C $\rightarrow$ c |
| | (2.5) | A $\rightarrow$ a B |

(1) S

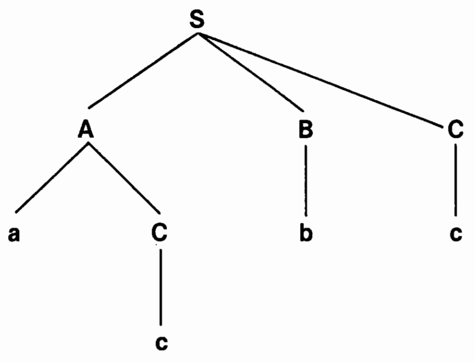
| | |
|---|---|
| A    B C | if you apply rule 2.1 |
| a C B C | if you apply rule 2.2 |
| a C b C | if you apply rule 2.3 |
| a c b c | if you apply rule 2.4 |

(2) S

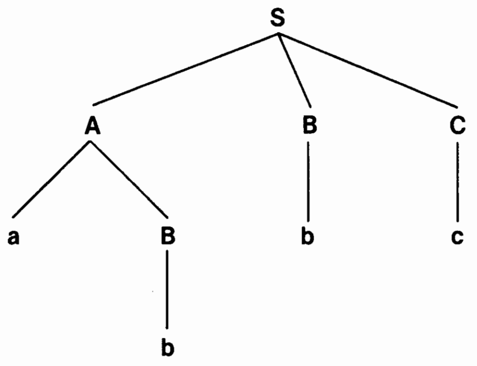
| | |
|---|---|
| A    B C | if you apply rule 2.1 |
| a B B C | if you apply rule 2.5 |
| a b b C | if you apply rule 2.3 |
| a b b c | if you apply rule 2.4 |

Sentence generation can best be represented in tree form, as the following example illustrates.

(1)



(2)



You can draw a tree by starting with the initial element S, then writing below it the sequence of elements that appears on the right side of a rule that begins with S. Next, draw branches from every element that can be expanded. A tree is complete when each branch ends with a terminal element. A grammar can generate a sentence only if all the branches of the tree that represents the grammar end in terminal elements.

## Abbreviatory Conventions

**4.3** Abbreviatory conventions combine a number of related grammar rules. If two grammar rules have the same rule element on the left side and the same first rule element on the right side, you can use one or more of the following abbreviatory conventions to combine the rules.

### Parentheses Convention

**4.3.1** If two grammar rules are identical except that one contains an element or a sequence of elements that the other does not, you can collapse the two rules by writing out the longer rule and enclosing the additional element(s) in parentheses. Consider the following pair of rules:

| | | |
|---|---|---|
| Grammar (3) | (3.1) | $A \rightarrow C\ D\ B$ |
| | (3.2) | $A \rightarrow C\ B$ |

The rules are the same except that the first contains one additional element. Collapse these two rules into a single rule by enclosing the additional element in parentheses:

(3.3) $A \rightarrow C\ (D)\ B$

This rule reads as follows:

The nonterminal element 'A' consists of 'C', followed by an optional element 'D', followed by 'B'.

### Braces Convention

**4.3.2** Braces indicate an either/or option when a rule can be expanded in more than one way. Consider the following rules:

| | | |
|---|---|---|
| Grammar (4) | (4.1) | $S \rightarrow A\ B$ |
| | (4.2) | $S \rightarrow A\ C$ |
| | (4.3) | $S \rightarrow A\ D$ |

The preceding rules state that a sentence consists of 'A', followed by 'B' or 'C' or 'D'. You can collapse these three rules into one by applying the braces convention:

(4.4) $S \rightarrow A\ \{\ B\ C\ D\ \}$

You can apply the parentheses and braces conventions together to further collapse a set of rules. For example, you can apply the parentheses convention to collapse rules 5.1 and 5.2 into a single rule (5.4):

|  |  |  |
|---|---|---|
| Grammar (5) | (5.1) | S ⟶ A B C |
|  | (5.2) | S ⟶ A C |
|  | (5.3) | S ⟶ A E |
|  |  | ↓ |
|  | (5.3) | S ⟶ A E |
|  | (5.4) | S ⟶ A (B) C |

But notice what happens if you try to apply the braces convention to collapse 5.3 and 5.4 into rule 5.5:

(5.5) S ⟶ A { (B) C E }

Rule 5.5 does not yet satisfy the criteria for collapsing all the rules (5.1 through 5.4), because it cannot generate 5.4. Another abbreviatory convention is required — the square brackets convention.

## Square Brackets Convention

**4.3.3** Square brackets define groups of rule elements inside braces. Consider the following two grammar rules:

|  |  |  |
|---|---|---|
| Grammar (6) | (6.1) | A ⟶ B C D |
|  | (6.2) | A ⟶ B E F |

You can collapse these two rules into a single rule with braces and square brackets. First, enclose the unshared elements within braces to indicate an either/or option.

(6.3) A ⟶ B { C D  E F}

Next, use square brackets within the braces to show that rule elements 'C' and 'D' constitute one group, while elements 'E' and 'F' constitute another.

(6.4) A ⟶ B { [C D] [E F] }

Rule 5.5 must be rewritten as 6.5 with square brackets, because the sequence '(B) C' constitutes one group:

(6.5) S ⟶ A { [(B) C] E }

These abbreviatory conventions are more than just a convenient shorthand for stating rules. They express important grammatical relationships and make computer processing of the grammar rules more efficient. These conventions save processing time and memory space for the Parser because they reduce the total number of rules that must be applied. However, you must be very careful when applying abbreviatory conventions. Suppose a grammar has the following two rules:

| | | |
|---|---|---|
| Grammar (7) | (7.1) | S → A B C D |
| | (7.2) | S → A E C F |

These rules state that either 'B' or 'E' follows 'A', and either 'D' or 'F' follows 'C'. At first glance, it appears that you can use braces here to collapse the two rules into a single rule:

| | |
|---|---|
| (7.3) | S → A { B E } C { D F } |

However, when you expand the S rule again, you generate not only the two original rules but also two rules not in the original grammar:

| | |
|---|---|
| (7.1) | S → A B C D |
| (7.2) | S → A E C F |
| (7.4) | S → A B C F |
| (7.5) | S → A E C D |

The correct way to combine rules 7.1 and 7.2 is:

| | |
|---|---|
| (7.6) | S → A { [B C D] [E C F] } |

Following are further examples of how abbreviatory conventions can result in meaningless rule expansions or inefficient computer processing.

- Inefficient use of abbreviatory conventions. The rule

  A → B ((C) (D) (E))

  can be simplified, with no loss of meaning, to

  A → B (C) (D) (E)

- Meaningless expansions. Consider the following grammar:

| | | |
|---|---|---|
| Grammar (8) | (8.1) | A → B |
| | (8.2) | A → B C |
| | (8.3) | A → B D |
| | (8.4) | A → B E |

Suppose you first use the parentheses convention to collapse rules 8.1 and 8.2, giving rule 8.5.

(8.5) $A \rightarrow B\ (C)$

Next, you use the braces convention to collapse rules 8.5, 8.3, and 8.4, giving rule 8.6.

(8.6) $A \rightarrow B\ \{(C)\ D\ E\}$

The braces convention indicates that one of the enclosed elements, and only one, must be chosen. An enclosed element in parentheses, as in rule 8.6, means that in one expansion of this rule no element is chosen. This is a meaningless use of the braces convention. If you wish to include the possibility that no element will be chosen in a certain rule expansion, make parentheses the outermost element, as in rule 8.7.

(8.7) $A \rightarrow B\ (\{C\ D\ E\})$

## NaturalLink Grammar

**4.4** The syntax of the NaturalLink grammar differs from standard context-free grammar notations. These differences serve to further enhance computer processing of the grammar. NaturalLink grammar rules must follow this format:

@<mother> <daughter1> . . . <daughterN>
;<translation list1>; ... <translation listM>!

where:

@ flags the beginning of a rule.

< mother > is the left side of the rule (that is, the rule element to the left of the arrow) in the context- free grammar format.

< daughter1 > is the first rule element on the right side of the rule (that is, the first rule element to the right of the arrow) in the context-free grammar format.

This element is also called the left corner. In the NaturalLink grammar format, this element must not be enclosed within braces, brackets, or parentheses.

< daughterN > are the remaining elements (if any) on the right side of the rule. Abbreviatory conventions can be used with these elements.

; flags the beginning of a translation list.

< translation list1 > and < translation listM > specify the translation list(s) for a rule (see paragraph 4.4.1, Translation Lists)

! marks the end of the rule and its translation list(s).

Rule elements in the NaturalLink grammar format, like those in the context-free format, are either terminal elements or nonterminal elements. They are separated by one or more spaces or carriage returns.

## Translation Lists

**4.4.1** Translation lists (the sets of numbers in parentheses that appear at the end of each grammar rule) tell the Translator which expansion of the rule was used in the parse and how to build the target language string. The numbers in the first set of parentheses are the syntactic path list. They describe how to form one expansion of that grammar rule. The numbers in the second set of parentheses are the semantic substitution list. They tell the Translator how to combine the individual translation strings to create the correct target-language command. The target language string for each terminal element in the grammar is contained in the lexicon.

So that it can tell which expansion of the grammar was used in the parse, the NaturalLink software assigns an integer to each rule element. The left side is assigned zero, and each succeeding element is incremented by one. The next examples follow the context-free grammar format for clarity of discussion. For example, the rule

```
X —> A B C
```

is assigned the values

```
X —>  A B C
0     1 2 3
```

and the translation list is assigned the values

```
;(1 2 3) (x1..xN)
```

The expansion of this rule is represented by the sequence of the integers (1 2 3) and (x1..xN). The first sequence of integers, (1 2 3), is the syntactic path list; it indicates which path is taken through the right side of the rule. The second set of integers, (x1..xN), is the semantic substitution list.

If the rule has more than one possible expansion, there will be more than one translation list. Following is an example of a rule with more than one possible expansion:

```
X —>  A  {[B  C]   [D  E] } F
0     1    2 3     4 5    6
```

The translation lists are

; (1 2 3 6) (x1...xN) ; (1 4 5 6) (y1...yN) !

In this example, the syntactic path list (1 2 3 6) is the path for the rule expansion X —> A B C F, and (1 4 5 6) is the rule expansion for X —> A D E F.

The semantic substitution list indicates to the Translator how it should build the target string for the rule. This list consists of the following parts:

- Predicate — The first integer of the list
- Arguments — The integers after the predicate
- Predicate's string — A character string associated with the predicate
- Slots — Locations in the predicate's string where the arguments are placed (in the order by which they occur in the semantic substitution list)

Each argument in the semantic substitution list also has a character string associated with it. These strings are stored in the lexicon and are the target language translations for the lexical items.

The slots in the predicate's string are indicated by an at sign (@) followed by an integer. The @1 indicates that the first argument is to be inserted in the @1 position, the @2 indicates that the second argument is to be inserted in the @2 position, and so on.

To develop the semantic substitution list, follow these steps:

1. For representative sentences in the language, draw the parse tree for each. The NaturalLink Toolkit automatically generates these sentences.

2. For each terminal node in a tree, determine the target language fragment that corresponds to that lexical item.

3. Decide which target language fragments you wish to use as predicate strings and which are to be argument strings.

4. Add the @< integer> slots to the predicate strings, and create the semantic substitution lists to reflect how you want these strings combined.

Refer to the *NaturalLink Technology Workbook* for an extended example of how to develop the semantic substitution list for a specific grammar.

As an example, assume that the rule

```
X →  A  {[B  C]   [D  E] }  F
0    1    2  3     4  5     6
```

has the following translation lists:

; (1 2 3 6) (2 3 1 6) ; (1 4 5 6) (4 5 1 6) !

The predicates are 2 and 4. Assume 2's predicate string is "@1 and @2 and @3", and 4's predicate string is "@1 or @2 or @3". If the rule expansion used is $X \rightarrow$ A B C F, the string built by the Translator is:

< C's string> and < A's string> and < F's string>

Similarly, if the rule expansion used is $X \rightarrow$ A D E F, the string built by the Translator would be:

< E's string> or < A's string> or < F's string>

Rule elements can also be repeated so that the same string can be inserted into several slots. For example, a valid predicate string is

@1 @1 little @2

If "twinkle" is the string of the first argument, and "star" is the string of the second argument, the final string is

twinkle twinkle little star

Nesting is allowed in the semantic substitution list and is designated by an additional set of parentheses. For example, the translation list

; (1 2 3 6) (2 (3 1) 6) !

tells the Translator to first use predicate 3's string and place daughter 1's argument string in the predicate's slot, and then to fill the slots of predicate 2's string with the string built from daughters 3 and 1 and with daughter 6's argument string.

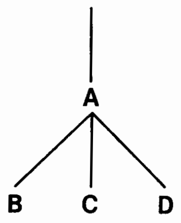
## Translation Process

**4.4.2** Understanding the translation process is important in constructing correct translation lists in the grammar (and correct translations for lexical items). See the *NaturalLink Technology Workbook* for examples of constructing translation information for a sample application.

The translation process works as follows:

1. The Translator traverses the parse tree and finds the lowest, rightmost subtree.

2. The Translator gets the translation list for the rule expansion that this subtree represents. For example, if the subtree

   is produced from the rule

   A $\longrightarrow$ B (C D) ; (1) (1) ;(1 2 3) (2 1 3) !

   the Translator determines that the proper translation list for this expansion is (1 2 3) (2 1 3).

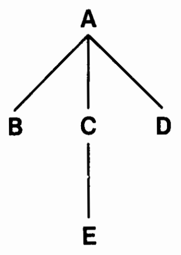
3. The Translator gets the predicate and makes the substitutions into the predicate string as prescribed in the semantic substitution list. In the example, the predicate is daughter 2. Daughter 2's string has two slots, @1 and @2. The Translator takes daughter 1's string and inserts it into the first slot in daughter 2's string; it then takes daughter 3's string and inserts it into the second slot.

4. The Translator moves up the tree to find the current subtree's mother, which is 'A'.

5. The Translator repeats steps 2 through 4 until the root of the parse tree is processed.

The preceding algorithm assumes a tidy set of translations and subtrees. Some special cases that can appear in an application require further explanation:

- If at any time the mother of a particular subtree has subtrees under it that have not been processed, that subtree is processed before the mother is processed. For example, if the subtree is

  then the subtree 'C', which has 'E' as a daughter, is processed before the whole.

- If all the slots in the predicate's string are not filled (that is, if the predicate's string has more slots than the number of arguments) in step 3, the remaining slots are reset relative to 1. Thus a string containing @3 and @4 would be reset to @1 and @2, as demonstrated in the following example:

  X ⟶ A B C ; (1 2 3) (1 2 3) !

The predicate in this rule is daughter 1, which has string "@1 and @2 and @3 and @4." In this case, the predicate has only two arguments, daughters 2 and 3. Therefore, @3 and @4, which have no arguments, are reset to @1 and @2, with the following result:

< B's string> and < C's string> and @1 and @2

Farther up in the tree, these slots must be replaced with other arguments; otherwise, the final translation will have unresolved slots.

- If some rule elements do not have translations associated with them, they do not appear in the semantic substitution list. In the following example, B does not have an associated translation:

  X $\rightarrow$ B C D ; (1 2 3) (2 3) !

- Experts should not occur as predicates with arguments in the semantic substitution list in any rules of the grammar. The reason is that experts cannot contain slots.

- If the semantic substitution list contains only a single element, the string that replaces that element is passed up the parse tree unchanged.

  A $\rightarrow$ B ; (1) (1) !

## How NaturalLink Uses the Grammar File

**4.5** The grammar file is a key component in the development of a NaturalLink interface. The grammar file provides input for the lexicon. Together, the lexicon and the grammar file contain the information that the Parser and the Translator need to manipulate the NaturalLink screen and to develop the target language string from the English sentence.

### Grammar File and Lexicon

**4.5.1** The following paragraphs explain in detail the relationship between the grammar file and the lexicon.

The lexicon defines the items listed on the screen and relates them to items in the grammar; it also relates the terminal elements in the grammar to the target language. Each lexical entry includes the following information:

- Terminal element in the grammar

- Screen position — Specified by the item text and the window in which it appears

- Translation — Either the predicate or argument string

The lexicon contains at least one entry for each terminal element in the grammar. Special cases can exist, as in the following examples:

- A terminal element can be associated with more than one English phrase in the NLmenu screen. In this situation, the lexicon must contain one entry for each English phrase corresponding to that terminal element. It is possible that these multiple entries have different translations.

- Several terminal elements in the grammar can map to the same screen position with each having the same or different translations. The Parser can determine by examining the grammar which terminal elements to add to the parse. Note that each terminal element and screen position must map to a single translation; otherwise, an ambiguity results that will not be detected.

For information on constructing the lexicon, refer to Chapter 8, Generating the NaturalLink Lexicon.

# CREATING AN NLMENU SCREEN 5

## Introduction

**5.1** This chapter explains how to design and create the interface screen (NLmenu screen) using the Screen Builder utility, an interactive software tool that enables you to design or modify windows. The Screen Builder utility is documented in the *NaturalLink Window Manager Reference Manual.* Before creating the NLmenu screen, you should read that manual and become familiar with building and manipulating screen files.

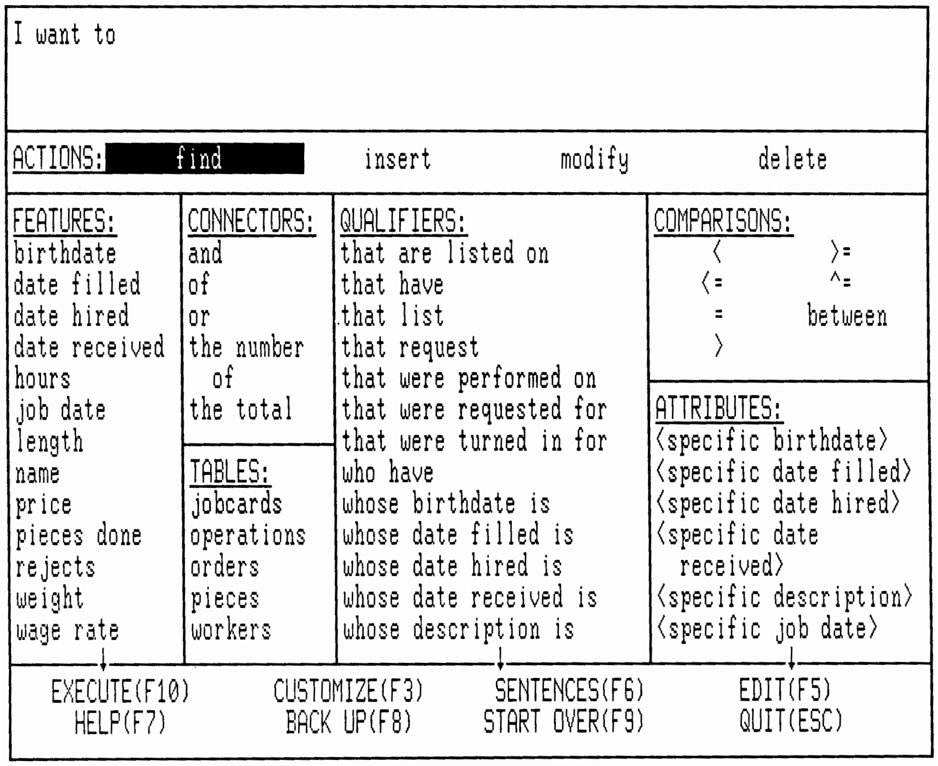
Assume that you are creating a NaturalLink interface that will access tables or files containing information about a company's inventory, personnel, production, operations, and so forth. The interface screen that you design will consist of a set of windows containing natural language words and phrases that the user can select to build the database queries or command sentences. During execution of the interface, the NaturalLink software translates a command sentence into the corresponding target language string and sends the translation and the command sentence to the application program.

The physical appearance of the NLmenu screen you design for a particular interface strongly influences a user's perception of the application. Therefore, windows and the information they contain should be presented in a logical and pleasing arrangement. This chapter discusses factors you should consider when designing the NLmenu screen, including the following:

- Types of windows needed
- Size of windows
- Placement of windows within a screen

Figure 5-1 illustrates a typical NLmenu screen for a NaturalLink interface to a relational database. The top window (with the label `I want to`) is the Results window. The seven windows displayed in the center of the screen (labeled `ACTIONS`, `FEATURES`, and so forth) are parse windows. The bottom window is the Command window.

**Figure 5-1** **NLmenu Screen**



```
I want to

ACTIONS:      find          insert          modify          delete

FEATURES:       CONNECTORS:   QUALIFIERS:                  COMPARISONS:
birthdate       and           that are listed on              <        >=
date filled     of            that have                       <=       ^=
date hired      or            that list                       =        between
date received   the number    that request                    >
hours             of          that were performed on
job date        the total     that were requested for      ATTRIBUTES:
length                        that were turned in for      <specific birthdate>
name            TABLES:       who have                     <specific date filled>
price           jobcards      whose birthdate is           <specific date hired>
pieces done     operations    whose date filled is         <specific date
rejects         orders        whose date hired is            received>
weight          pieces        whose date received is       <specific description>
wage rate       workers       whose description is         <specific job date>

   EXECUTE(F10)      CUSTOMIZE(F3)      SENTENCES(F6)      EDIT(F5)
    HELP(F7)          BACK UP(F8)       START OVER(F9)     QUIT(ESC)
```

## Development Process

**5.2** An NLmenu screen usually consists of the following windows:

- Command window (the bottom window in Figure 5-1) — Contains one or more of the nine NaturalLink command options. Using Screen Builder, you can select the following attributes for the command options in the window:

  - Visible or invisible
  - Selectable or unselectable

  You can also designate the window as a pop-up window. The NaturalLink software does not display a Command window designated as a pop-up window.

- Results window (the top window in Figure 5-1) — Displays the command sentence as the user builds it. You can design this window to:

  - Be displayed at all times and, optionally, show a flag indicating when a command sentence is complete (executable)

  - Appear only when the command sentence is executable or when the user selects the NaturalLink option Edit Experts to modify specific values in the command sentence

  - Not appear (a pop-up window that never appears)

  Do not use the last option if you have also designated the Command window as a pop-up window, or the user will not be informed when the command sentence under construction is executable.

- Parse windows — Contain the words and phrases the user selects to build NaturalLink command sentences. The parse windows in Figure 5-1 are:

  - ACTIONS
  - FEATURES
  - CONNECTORS
  - TABLES
  - QUALIFIERS
  - COMPARISONS
  - ATTRIBUTES

- Special windows — Includes windows not used by the NaturalLink Sessioner, Parser, or Translator, such as those designed only to display titles or present descriptive information.

## Command Window

**5.2.1** This window contains the nine NaturalLink command options. Only one of the nine options—Execute(**F10**)—is required and must be selectable. You can include some or all of the other eight command options in your NLmenu screen, depending on application requirements, or you can define several of your own command options and function keys.

Following is a list of the NaturalLink-supported command options and their associated default function keys. See Chapter 15 (The NaturalLink Sessioner, paragraphs 15.1 through 15.3) for additional information about the way the NaturalLink software handles the command options.

- Execute/**F10** — Causes the command sentence and its translation to be sent to the application program.
- Customize/**F3** — Allows the user to modify the phrases on the interface screen.
- Show/**F4** — Calls a pop-up window that displays the target-language translation string of the current command sentence.
- Edit/**F5** — Enables the user to edit the expert item(s) in the current command sentence.
- Sentences/**F6** — Calls a pop-up window that contains a subset of one or more of the following commands (depending upon which commands in the subset are currently valid):
  - Save a command sentence
  - Recall a command sentence
  - Delete a command sentence
- Help/**F7** — Calls Help message(s) to explain items or commands in the NLmenu screen. This option works in two ways, depending on whether it is invoked by the **F7** key or selected as an item:
  - Pressing the **F7** key displays Help message(s) available for the item the cursor is on.
  - Placing the cursor on the Help/**F7** option in the command window and pressing the **ENTER** key displays a message that explains how to use the Help/**F7** option.
- Back Up/**F8** — Erases the previous item or expert selection from the command sentence being built and returns the cursor to the previous active window.

- Start Over/**F9** — Erases the current command sentence and enables construction of a new NaturalLink command.

- Quit/**ESC** — Ends the current NaturalLink session and returns to the application program.

The NaturalLink software requires that a Command window be specified for every NLmenu screen. You design the Command window with Screen Builder, but you must designate the window as a Command window through the Lexicon Builder utility for the Sessioner to recognize it (see Chapter 8, Generating the NaturalLink Lexicon). The exact wording of the items in the Command window can differ from the wording shown in the example NLmenu screen (you could use Run instead of Execute, Stop instead of Quit, and so on). The order can differ as well. The Command window should be placed away from the other windows in the screen so it is not confused with the parse windows. A position at the very top or bottom of the screen is suggested.

Although the Command window is a required window, it need not be displayed on the NLmenu screen in every application. For example, if the application is to use only function keys for the command options, then the Command window does not need to appear on the NLmenu screen. You can make both the Command window and the required Execute command invisible by specifying the Command window as a pop-up window. In this case, you could use the Results window to inform the user when the command sentence is executable. (Refer to paragraph 5.2.2.1, Showing the Results Window, for more information about setting the Results window Execute flag.)

It is best that the Command window be visible and all the items (commands) in the window be selectable. A user can then invoke a NaturalLink command either by selecting the appropriate command option or by pressing its default function key.

## Results Window

**5.2.2** The Results window (the top window containing the `I want to` label in Figure 5-1) shows the user the current command sentence as it is being built. Like the Command window, the Results window is required by the NaturalLink software, and you must designate it using the Lexicon Builder utility. However, if you do not want this window to be displayed, you can design it as a pop-up window that does not appear on the NLmenu screen.

***Showing the Results Window***

**5.2.2.1** In most applications, the Results window should be displayed at all times. If you intend to display this window, it is best to specify it as a free-format display window. Then, as items are placed in the window, the sentence follows a natural format (that is, left to right, line by line).

Position the Results window on the screen so that it can be seen easily and is not confused with other windows. Market research has demonstrated that the user's attention is drawn to the top of the screen; this is the logical place to position the Results window. The use of different colors or intensities also helps differentiate this window from the others.

The possible length of the sentences determines the size of the Results window. Typically, room for three or four lines of text is sufficient. If the window is a free-format display window, and if the text has too many lines to fit within the window's horizontal boundaries, set the Show Last Item When Painted attribute to Yes (1) using Screen Builder. The text then scrolls automatically to show the last item selected. The user can scroll text manually if the Results window is a list or text window.

You can instruct the software to signal the user, by displaying a flag in the Results window, when the command sentence under construction becomes executable. Do this by specifying a title such as Run or Execute as the window label and by accepting the default value of No (0) for the Invisible Label attribute in Screen Builder. This label is automatically displayed when the command sentence is complete. If you do not want this flag to appear, set the value for the Invisible Label attribute to Yes (1). Since the NaturalLink software requires that the Results window be specified, you must specify some text for the Results window label even if it will be invisible.

Because the window label for the Results window can be a flag, it cannot identify the window to the user. You can create an identifying tag when you build the window by specifying a word or phrase, such as "I want to" (as in Figure 5-1), "Results," "Find," and so forth, as a window item. Items specified in the Results window are never removed.

***Not Showing the Results Window***

**5.2.2.2** In some applications, you may prefer not to display the Results window at all times. You can accomplish this by specifying the Results window as a pop-up window. In this case, you have the following options:

- Display Pop-up When Command is Executable/Editable — If you choose this option, set the window's priority to a nonzero value with Screen Builder. The Results window then pops up to cue the user when the command can be executed or edited.
- Never Display Pop-Up — If you choose this option, accept the default window priority of 0 (zero) in Screen Builder. Do not choose this option if you have also specified the Command window as a pop-up window, because if you do there will be no way to signal the user when the command sentence is complete.

## Parse Windows

**5.2.3** The parse windows contain the words and phrases a user selects to build a NaturalLink command sentence. The NaturalLink software makes these windows either active or inactive, depending upon which of them contains the next logical item(s) to be selected.

### *Designing Parse Windows*

**5.2.3.1** Following are some guidelines for designing these windows with Screen Builder.

1. Decide what phrases to use in building the sentences. The specific categories used in the grammar are your guide.

2. Arrange these phrases in logically related groups. Note the groups in Figure 5-1: `ACTIONS`, `FEATURES`, `CONNECTORS`, `TABLES`, `QUALIFIERS`, `COMPARISONS`, and `ATTRIBUTES`. Try not to group phrases in the same window if they immediately follow each other in a sentence.

3. Determine how wide the window should be to contain the group, using the longest phrase in each group as a criterion. Determine the length of the window from the number of items in the group. Because the user can scroll, the window does not have to be long enough to show all the items. However, you should show as many as possible, because the ability to look ahead for choices is an important factor in an application's ease of use. Vertical window borders take up one character of space each, and horizontal borders take up one line each; be sure to add this space to your calculations if you are using a bordered window.

   Also remember that all the items may not be visible at the same time. When a window is active, NaturalLink shows only the valid choices. Thus, a smaller window may serve. For example, if no more than 5 out of 10 items will ever appear at one time as valid choices, the window need only be large enough to display 5 items.

4. Position the windows on the screen. It is best to place the windows on the screen in the order that the phrases are to be chosen during sentence construction. This allows the user to move from left to right and from top to bottom through the windows.

5. When designing the screen with Screen Builder, you may wish to place words or phrases in the parse windows so that you can see how the NLmenu screen will look after the interface is built. However, this is not required, as you can specify these phrases when you build the lexicon (see Chapter 8, Generating the NaturalLink Lexicon).

***Saving Screen Space*** **5.2.3.2** If screen space becomes too limited, try one of the following remedies:

- Make the windows smaller by using multiple-line items in single-column windows. (Refer to the CONNECTORS and ATTRIBUTES windows in Figure 5-1 for examples.) To do this, set the window format Disable Multiple Line Items? attribute to No (0) in Screen Builder.
- Use multiple-column windows for short items. (Refer to the COMPARISONS and ACTIONS windows in Figure 5-1.)
- Use an expert window for selection from a list of specific items or for an item that changes frequently. You define expert windows through the Generate or Modify the Lexicon option in the Interface Builder utility. For more information about expert windows, refer to Chapters 8, Generating the NaturalLink Lexicon, and 9, Application Control of User Options.
- Combine two or more windows into one. If items do not appear at the same time when the window is active, or if items do not follow each other in a sentence, they can be grouped together by semantic or syntactic characteristics. For example, the CONNECTORS window has and, of, and or grouped with the number of and the total. These two sets of items do not perform the same function, but they are never selectable consecutively. They are thus good candidates to place in the same window.
- Specify the Command window as a pop-up window if it is seldom used in building command sentences. Then the Sessioner can add and delete it as necessary.

## Special Windows

**5.2.4** The NLmenu screen can contain windows other than parse, Command, and Results windows. These other windows can provide special bordering effects or display titles and other descriptive information. However, be careful not to clutter the screen or distract the user with unnecessary visual elements. Typically, special windows are display windows whose usual purpose is to present additional information. They are displayed by the Sessioner (if they are not pop-up windows) along with other NLmenu windows, but the user cannot make selections from them and the application program cannot manipulate them.

## Setting Window Attributes

**5.3** The overall look of the NLmenu screen depends on individual application and design requirements. However, several attributes for NLmenu windows must be set:

- Windows from which items are selected (Command and parse windows, for example) must be list windows or user-defined windows.

- Parse windows cannot be multiple-selection windows.

- The Results window can be a text, display, list, or user-defined window.

- The NaturalLink Interface Builder requires that all NLmenu windows have window labels. However, the labels need not be displayed in the windows. To prevent the labels from being displayed, set the Invisible Label attribute when you define windows with Screen Builder.

- During NaturalLink processing, windows that contain selectable items are active. To differentiate between active and inactive windows during display, Screen Builder sets the intensity level of active windows to 7 (white) and of inactive windows to 4 (green). You can change these intensities.

- Do not set the Special Repaint on Receive attribute for any window that appears in your NLmenu screen. The reason is that when the receive call is made on the window with the Special Repaint on Receive attribute, other windows in the NLmenu screen are not painted.

- If several windows in an NLmenu screen are made active at the same time during the sentence-building process, the NaturalLink Sessioner places the cursor in the window closest to the top of the screen, moving from left to right. This is the default pattern for the Sessioner. If you want the cursor to be placed in a different window, give that window a higher priority than the other windows that will be active at the same time. The Sessioner then places the cursor in the window with the highest priority. If all the windows have the same priority, the Sessioner places the cursor in the window closest to the top.

## Window Order

**5.4** The order of the windows in the NLmenu screen is not important. However, when the interface file is generated, the windows are reordered as follows:

1. All parse windows come first (same order as in the NLmenu screen, but moved to the top)
2. Command window
3. Results window
4. All other windows (same order as in the NLmenu screen, but moved to the bottom)

When you create the NLmenu screen, you may find it helpful to put the windows in this order; then the order of the windows in the NLmenu screen file and the interface file will be the same.

The screen number of the NLmenu screen to be used in Window Manager callable routines is calculated by adding 20 to the interface number. The interface number is determined when the interface is loaded (see Chapter 15, The NaturalLink Sessioner). The window number to be used in any call must be the number of the window in the interface file, not in the NLmenu screen file.

## User-Defined Windows in the NLmenu Screen

**5.5** Any of the windows that comprise an NLmenu screen—the Command window, the Results window, parse windows, and special windows—can be user-defined. You create a user-defined window (UDW) for an NLmenu screen the same as other windows. For example, a UDW used as a Command window must have a window label and an Execute item.

A UDW used as a parse window must contain the phrases for all lexical items assigned to it, as a list window would. All the same restrictions for window attributes also apply. The application must control the display of the items in the window; the Parser merely sets the item attribute to visible or invisible.

One example of when it might be better to use a UDW (instead of a list window) as a parse window is in auto-selection of expert items. If the only item available for selection during the sentence-building process is an expert item and the sentence is not executable, the Sessioner will automatically select the item. This means that the window containing the expert item is repainted with only the expert visible, but the user is not required to select that item; the window to elicit the expert value is displayed automatically. This repainting of the window that contains the expert item can confuse the user. To avoid this confusion, assign to a UDW all expert items that will automatically be selected. When Window Manager calls your application display routine to display the UDW containing your expert items, return a 0 (zero). The Sessioner will automatically select the expert item and continue processing as expected.

**CAUTION: If you use this procedure to handle auto-selection of experts, be sure that only one expert item is available for selection at any time. If more than one is available, the system could hang.**

For more information on UDWs, refer to Chapter 2, Window Manager Features, and Chapter 8, User-Defined Windows, in the *Window Manager Reference Manual.*

# USING THE NATURALLINK INTERFACE BUILDER 6

## Introduction

**6.1** The NaturalLink Interface Builder (IBUILD) utility provides an automated means of building and testing NaturalLink interfaces. This is an important capability because the NaturalLink Parser accepts only error-free input from the interface, thus utilizing the computer's processing power and internal memory more efficiently than do many other natural language systems.

The Interface Builder utility maintains a history of all interfaces currently under development. Associated with each interface are grammar, lexicon, expert information, and screen description files. As an interface is built, tested, or modified, the status of its components is recorded in the history file. Because the NaturalLink software keeps track of interface development, it is able to guide you through the development process and to eliminate errors that may occur if you attempt to generate an untested interface.

## Procedure

**6.2** The following paragraphs explain the Interface Builder utility.

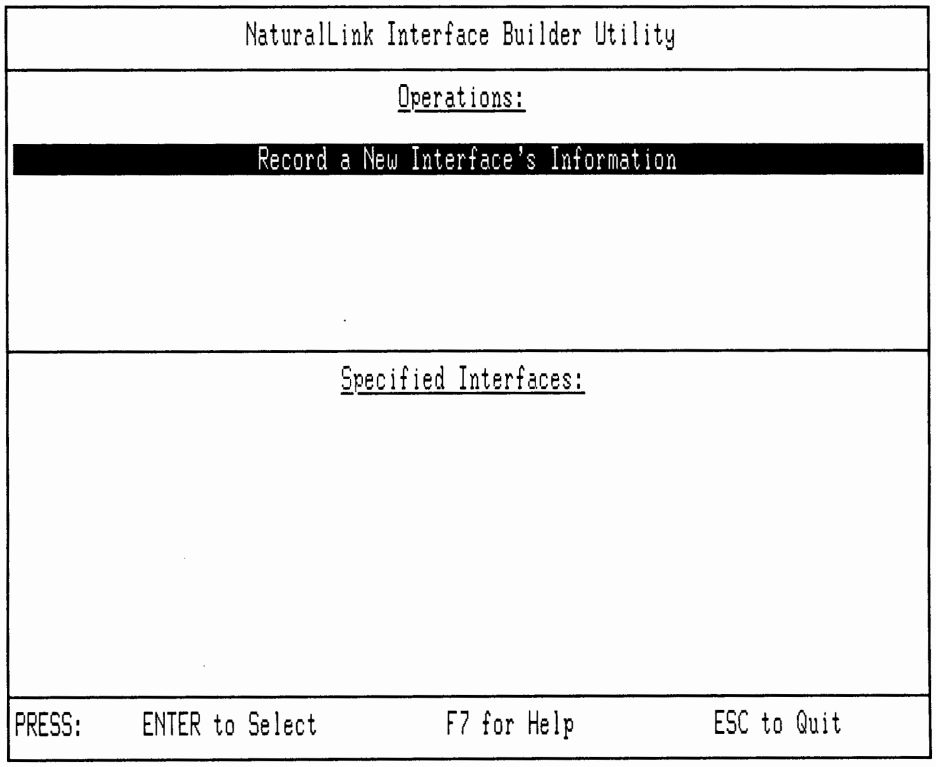
### Setting Environment Variables

**6.2.1** Before entering the Interface Builder utility, you should first set up two environment variables with the MS-DOS SET command. These variables are NLXTOOLS and NLXHIST. NLXTOOLS must be set to the directory pathname in which the screen files and message files for the IBUILD utility exist. If NLXTOOLS is not set, IBUILD will expect these support files (the .PIC, .NS$, and .NM$ files) to be in the default drive and directory. NLXHIST must be set to the directory pathname in which IBUILD will create and maintain the history file. If NLXHIST is not set, IBUILD will create the history file in the default drive and directory. If you are using a version of MS-DOS that does not support directories, ignore NLXTOOLS and NLXHIST.

### Starting Interface Builder

**6.2.2** You invoke the Interface Builder by typing IBUILD. If no interfaces have been previously specified (that is, if no interfaces are currently under development), the screen depicted in Figure 6-1 appears.

**Figure 6-1** **Record a New Interface's Information**



```
                     NaturalLink Interface Builder Utility

                                  Operations:

                    Record a New Interface's Information


                              Specified Interfaces:


PRESS:     ENTER to Select           F7 for Help           ESC to Quit
```

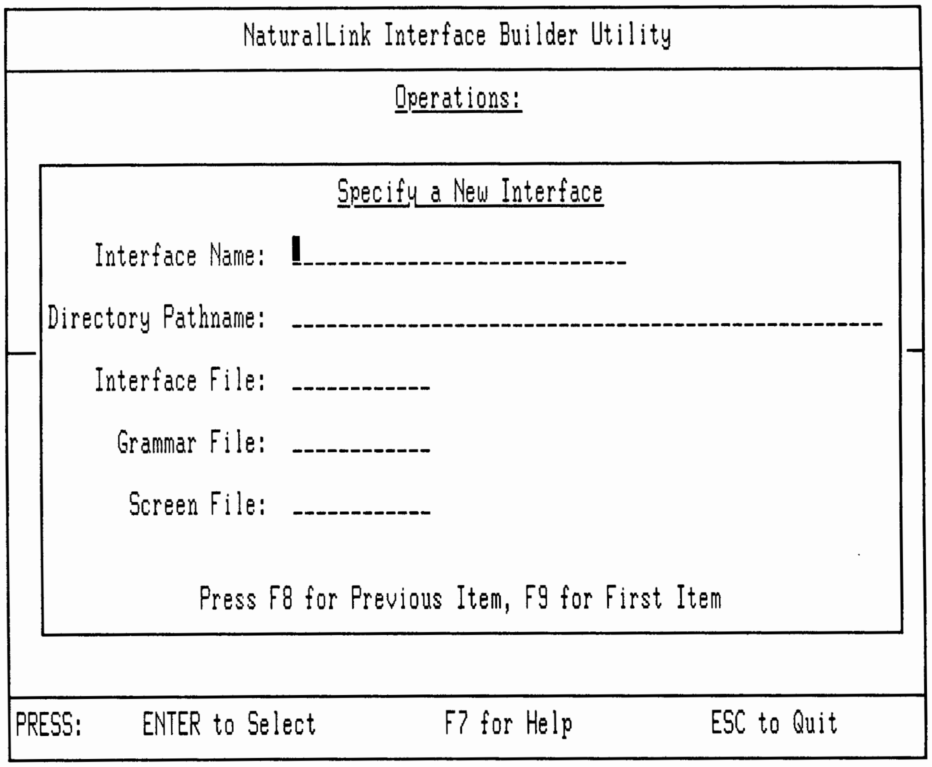
When this screen appears, the cursor is on the Record a New Interface's Information option in the Operations window. Select this option by pressing the **ENTER** key. A pop-up window (Figure 6-2) appears in which you can enter the necessary data for the new interface.

---

**NOTE:** Refer to Appendix G in the *NaturalLink Window Manager Reference Manual* for information about function keys on specific machines.

---

**Figure 6-2** **Specify a New Interface**



```
                  NaturalLink Interface Builder Utility

                            Operations:

                      Specify a New Interface

      Interface Name: ___________________________

 Directory Pathname: _________________________________________________

      Interface File: ____________

        Grammar File: ____________

         Screen File: ____________

              Press F8 for Previous Item, F9 for First Item

PRESS:     ENTER to Select          F7 for Help          ESC to Quit
```

You are prompted for the following information:

- Interface Name — A unique name that is entered in the history file and used to refer to the interface under development.
- Directory Pathname — The pathname of the directory in which all the files for this specific interface reside.

  This pathname is optional. However, if you leave it blank and then attempt to execute IBUILD from another directory (other than the one where the interface's files are located), the interface will not be available. With the aid of the NLXTOOLS and NLXHIST environment variables, the directory pathname, and the MS-DOS PATH variable, you can execute IBUILD from any drive or directory on your system; thus, you are not restricted to any one directory or forced to change directories repeatedly.

If you are using an operating system that does not support a hierarchical directory structure, you should either leave the directory pathname blank or enter a drive designator.

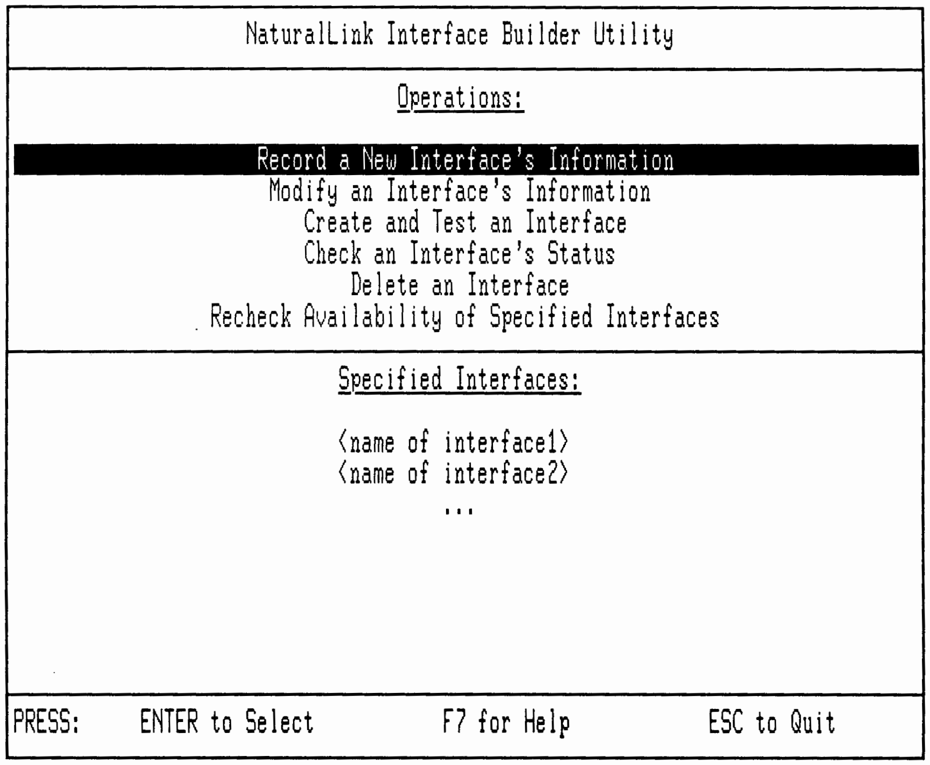
- Interface File — The output binary interface file that the application program will use. Current operating system file-naming conventions must be followed. The interface filename without the extension is used for system-generated files (lexicon file, expert information file, and sentence file).
- Grammar File — The input grammar file.
- Screen File — The input screen description. This file is created with the aid of the Screen Builder.

It is not necessary to enter a pathname or drive designator for the interface, grammar, or screen files. If a drive designator is entered, it will be ignored. The software locates these files through the directory pathname or, if it is blank, in the current directory.

The grammar file and the screen file must be in the directory named by Directory Pathname. After you specify the grammar file, a File Not Found error message is displayed if the file is not in that directory.

Although the grammar file is the only file that must already exist before you can record information about a new interface, you must respond to all filename prompts before you can proceed to the next option in the Operations window. After you enter this information, the NaturalLink software adds an entry in the history file for the new interface. The main NaturalLink Interface Builder utility screen (Figure 6-3) is then displayed.

**Figure 6-3** **Main Interface Builder Screen**



```
                 NaturalLink Interface Builder Utility

                             Operations:

               Record a New Interface's Information
                 Modify an Interface's Information
                     Create and Test an Interface
                       Check an Interface's Status
                          Delete an Interface
              Recheck Availability of Specified Interfaces

                        Specified Interfaces:

                         <name of interface1>
                         <name of interface2>
                                 ...

PRESS:     ENTER to Select          F7 for Help          ESC to Quit
```

There are two parts to this screen:

- The Operations window lists the various operations that can be performed on an interface.
- The Specified Interfaces window lists the interfaces currently under development.

In addition to Record a New Interface's Information, the following operations or options are available from this screen:

- Modify an Interface's Information. This option enables you to specify different values for an interface in the Record a New Interface's Information pop-up window.

- Create and Test an Interface. This option enables you to select an interface for testing from the list of interfaces under development (that is, the interfaces listed in the Specified Interfaces window). You can perform the following operations on the selected interface:

  - Test the Grammar for the Interface
  - Generate or Modify the Lexicon for the Interface
  - List the Lexicon for the Interface
  - Generate Representative Sentences for the Interface
  - Check the Screen Against the Lexicon for the Interface
  - View the Screen for the Interface
  - Specify Help and Window Coordinates for Sessioner Windows
  - Generate the Interface File
  - Test the Interface's Translations

  If a screen file does not exist, many of the options listed above will not be available in the Create and Test an Interface screen. For example, the View the Screen option and the Generate or Modify the Lexicon for the Interface option will not be available.

Each of these options represents a phase in the Create and Test an Interface procedure, but not all of the options will be available for selection at any given step in the procedure. Some options appear only after the execution of previous phases upon which they depend. For example, the lexicon cannot be generated until the grammar has been tested. Likewise, the interface file cannot be generated until the lexicon is complete and the screen has been checked against the lexicon. Chapters 7, 8, 10, 11, 12, 13, and 14 describe each phase of the Create and Test an Interface option in greater detail.

- Check an Interface's Status. This option displays the information in a selected interface's history file. It lists the directory pathname of the selected interface, the names of existing files, the date and time each file was last modified, and the status of each test. You can also use the **F6** key in the Create and Test an Interface utility to check an interface's status.

- Delete an Interface. This option removes an interface from the list of interfaces under development in the history file. No files associated with the interface are deleted, however.

- Recheck Availability of Specified Interfaces. This option determines the currently available interfaces and updates the Specified Interfaces window accordingly. The software performs this check each time the Interface Builder is invoked. You can also select this option if you need to recheck interface availability. For example, if you invoke the Interface Builder with interfaces previously specified to be on a particular diskette but you have the wrong diskette in the drive, the interfaces will not be available. To correct this, insert the correct diskette and select Recheck Availability of Specified Interfaces.

# TESTING THE NATURALLINK GRAMMAR 7

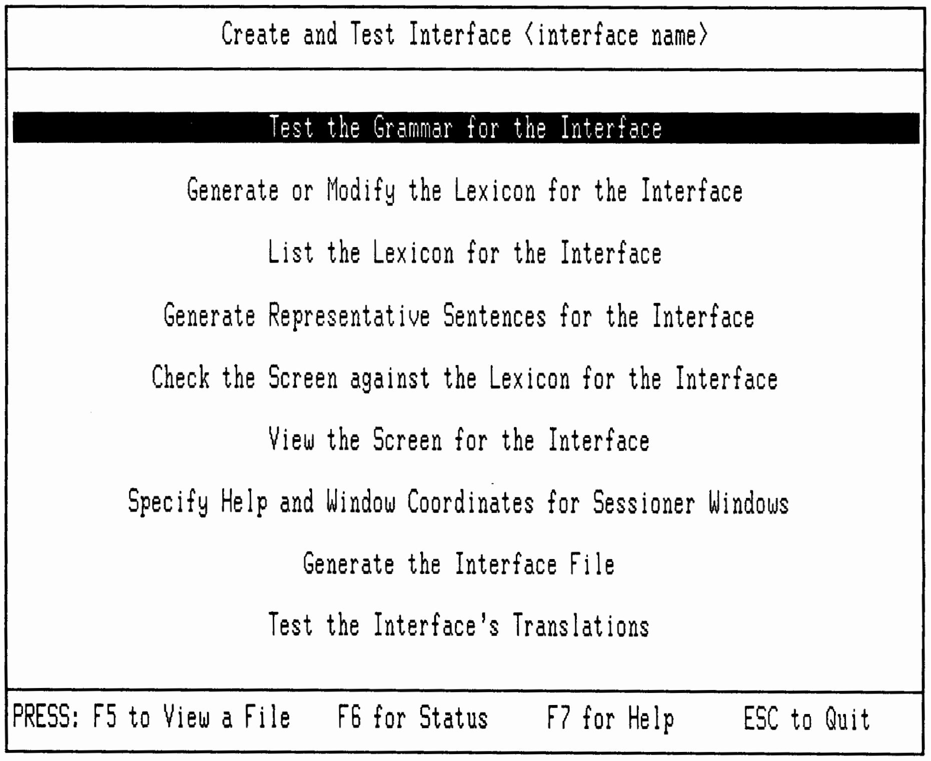
## Introduction

**7.1** After recording the interface information, test the interface grammar. From the Operations window of the main Interface Builder screen, select the Create and Test an Interface option. The cursor moves to the Specified Interfaces window. If you have specified more than one interface, select the interface whose grammar you wish to test by moving to its name and pressing **ENTER**.

A screen similar to the one shown in Figure 7-1 appears. The options available on this screen will differ from those shown, depending on the state of the interface.

When you are using the Create and Test Interface <interface name> screen, you can check the current interface status by pressing the **F6** key. The status listing will show the directory pathname of the current interface, the name of current files, the size of each file, the date and time each file was last modified, and the status of each test.

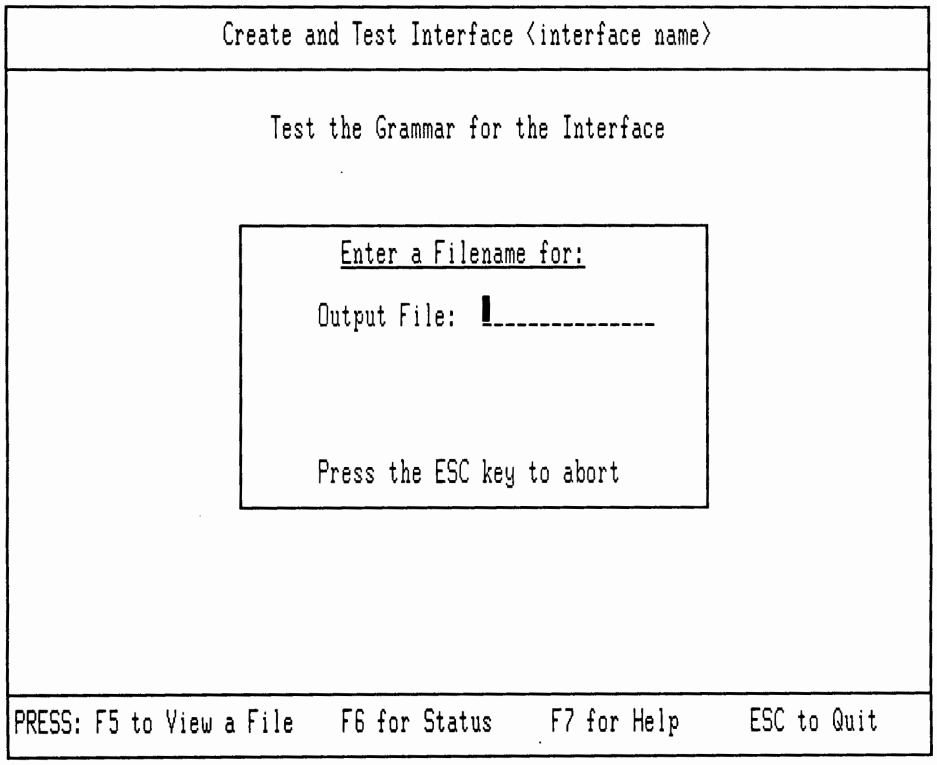
**Figure 7-1** **Create and Test an Interface**



```
Create and Test Interface <interface name>

Test the Grammar for the Interface

Generate or Modify the Lexicon for the Interface

List the Lexicon for the Interface

Generate Representative Sentences for the Interface

Check the Screen against the Lexicon for the Interface

View the Screen for the Interface

Specify Help and Window Coordinates for Sessioner Windows

Generate the Interface File

Test the Interface's Translations

PRESS: F5 to View a File    F6 for Status    F7 for Help    ESC to Quit
```

If you select Test the Grammar for the Interface, a pop-up window prompts for an output filename for the grammar test results (Figure 7-2). The output file you name is created in the directory specified by the interface's directory pathname.

**Figure 7-2** **Filename Pop-Up Window**

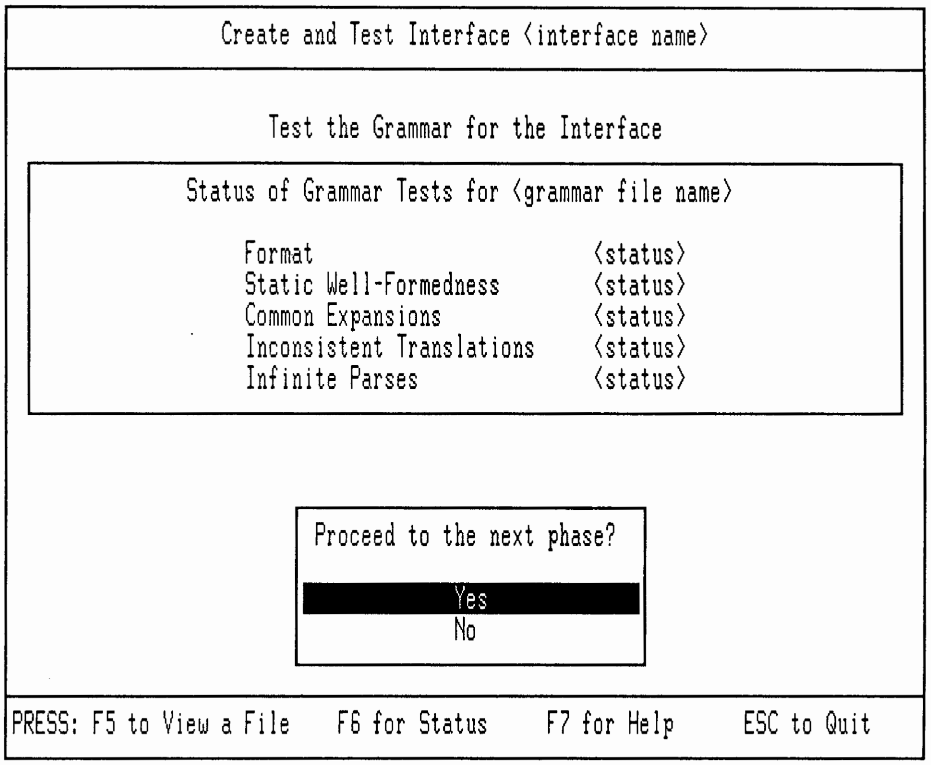


```
Create and Test Interface <interface name>

Test the Grammar for the Interface

Enter a Filename for:

Output File: ______________

Press the ESC key to abort

PRESS: F5 to View a File    F6 for Status    F7 for Help    ESC to Quit
```

If this file already exists, you have two options: (1) append the output to the contents of the current file, or (2) replace the current file.

After you supply the output filename, the screen shown in Figure 7-3 appears.

**Figure 7-3** **Grammar Test Pop-Up Window**



```
                 Create and Test Interface <interface name>

                   Test the Grammar for the Interface

           Status of Grammar Tests for <grammar file name>

               Format                          <status>
               Static Well-Formedness          <status>
               Common Expansions               <status>
               Inconsistent Translations       <status>
               Infinite Parses                 <status>

                      Proceed to the next phase?

                                 Yes
                                 No

PRESS: F5 to View a File    F6 for Status    F7 for Help    ESC to Quit
```

The Test the Grammar for the Interface option consists of five parts: the Format Test, the Static Well-Formedness Test, the Common Expansions Test, the Inconsistent Translations Test, and the Infinite Parses Test.

Here are brief descriptions of each:

- Format Test — Checks that only valid characters appear in the grammar and that there are no unbalanced brackets, braces, parentheses, or other syntax errors.
- Static Well-Formedness Test — Checks that all left sides of the grammar rules can be reached. It also tests whether all terminal elements in the grammar have corresponding lexical items. If they do not, it reports that the lexicon must be updated.
- Common Expansions Test — Checks for rules that generate a duplicate path when all abbreviatory symbols are expanded.

- Inconsistent Translations Test — Checks whether each possible path through a grammar rule has its own translation list and whether there are any extra translation lists.

- Infinite Parses Test — Checks whether any sets of rules in the grammar will lead to an infinitely looping path through the rules.

After each test, you can quit by responding negatively to the "Proceed to next phase, Yes/No?" question.

The < status> entry beside each test entry in the window has one of the following values:

- Not yet run. The test has not been run during this "Test the Grammar" session.

- Running. The test is in progress.

- Succeeded. The test has successfully completed.

- Failed. The test encountered a fatal error.

- Update Lexicon. The Static Well-Formedness Test returns this status if the lexicon does not exist, is not yet complete, or does not agree with the grammar.

If you are working with an especially large grammar, some of the grammar tests may take several minutes to complete.

## Errors Reported by Each Test

**7.2** Following is a discussion of the errors reported by each grammar test and output to the specified output file. The listings of errors reported do not include recommended remedial action. For this information, refer to Appendix B, NaturalLink Error Codes.

### Format Test

**7.2.1** The Format Test reports an error if it finds invalid characters in any area of a grammar rule. The valid characters in a grammar rule are:

@category_chars;translation_chars!

| | |
|---|---|
| category_chars | consists of letters, integers, spaces, underscores, carriage returns, parentheses, braces, square brackets, and slashes. |
| translation_chars | consists of integers, spaces, carriage returns, semicolons, and parentheses. |

If the Format Test encounters an error in the grammar, subsequent tests cannot proceed. The types of errors that can occur are:

- No '!' terminating previous rule.
- No '@' flag at the beginning of the rule.
- Translation list missing.
- Too many elements in the rule (the limit is 10 elements per rule). Elements include only terminal and nonterminal symbols in the grammar. Abbreviatory symbols are not considered elements.
- Not enough elements in the rule (for example, only one element; the right side is missing).
- Unbalanced abbreviatory symbol.
- Invalid character encountered.
- Complex left corner encountered (an abbreviatory symbol encountered as the first element on the right side of the grammar rule).
- Parenthesis immediately inside a brace. This produces an invalid expansion of the brace. According to grammar rules, one element within the brace must be chosen; an item enclosed in parentheses allows the possibility that none can be chosen.
- Integer in translation list greater than the number of elements in the rule.
- Second set of integers in a translation list missing.
- Too many levels of nesting in the rule (the limit is 10 elements per rule)
- Braces immediately inside a brace (redundant construct).
- All items optional within an abbreviatory construct.
- Missing translation list flag ';'.
- Nonintegers in the first sequence of integers in the translation list.
- Error encountered in closing the file.

For the Format Test to complete successfully, you must have a valid translation list for each grammar rule. You may want your grammar to pass the Format Test before you determine the translation lists, so that you can see if the grammar will pass other tests (the Common Expansions and Infinite Parses tests, for example). To make your grammar pass the Format Test, you can enter '(1)(1)' as the translation list for each grammar rule. Note that the Inconsistent Translations Test will report errors when you do this.

## Static Well-Formedness Test

**7.2.2** The Static Well-Formedness Test will report an error under the following conditions:

- There is an unreachable left side (other than the start symbol).
- All terminal elements in the grammar are not represented in the lexicon.
- All lexical entries are not assigned to both a window and a phrase.
- There is no lexicon file (you have not created one, or it has been deleted).

Unreachable left sides are elements in the grammar that appear only on the left side of a rule, never on the right side. If the Static Well-Formedness Test encounters any unreachable left sides other than the start symbol, the grammar test will not proceed.

The start symbol is the symbol that appears on the left side of the *first* grammar rule. Frequently an "s" (for sentence) is the start symbol, but it may be any symbol. The start symbol is usually unreachable and does not generate an error.

If the test encounters any of the conditions in the last three items listed above, it simply tells you to update your lexicon. You will have to determine the cause of the error and correct the condition.

## 7.2.3 Common Expansions Test

The Common Expansions Test reports an error if any rules generate the same path through the grammar. The following rules are an example:

Rule A: $S \rightarrow A$ {B C} {D E};

which expands to:

```
1. S  -->  A  B  D;
2. S  -->  A  B  E;
3. S  -->  A  C  D;
4. S  -->  A  C  E;
```

Rule B: $S \rightarrow A$ {Q C} {D F};

which expands to:

```
1. S  -->  A  Q  D;
2. S  -->  A  Q  F;
3. S  -->  A  C  D; Same as Rule A, expansion 3
4. S  -->  A  C  F;
```

Identical paths through the grammar are wasteful for the Parser. Identical paths can also affect the operation of the Infinite Parses Test. If the Common Expansions Test encounters identical paths, the grammar tests will not proceed. The Common Expansions Test reports an error of this type: Rules with Common Expansions.

## 7.2.4 Inconsistent Translations Test

The Inconsistent Translations Test reports an error if any paths through the grammar rules do not have translation lists or have extra translation lists. For example:

Element # 's:

```
@S A {B C}  {D  E} ;(1 2 4)(1 2 4);(1 2 5)(1 2 5)!
 0 1  2 3    4  5
```

The following expansions have translations lists:

$S \rightarrow$ A B D (1 2 4)

and

$S \rightarrow$ A B E (1 2 5)

The following expansions do not have translation lists:

S ⟶ A C D

and

S ⟶ A C E

Missing or extra translation lists will adversely affect the Interface Builder utility, the Parser, and the Translator. Errors encountered by the Inconsistent Translations Test must be fixed before the interface is built but will not affect the rest of the grammar tests. The types of errors reported by this test are:

- Grammar rule paths with no matching translation lists
- Translation lists with no matching grammar rule path

## Infinite Parses Test

**7.2.5** The Infinite Parses Test reports an error if any set of rules in the grammar can produce an infinitely looping path. For example:

S ⟶ A B C;
A ⟶ B;
B ⟶ A;
B ⟶ b;
C ⟶ c;

The path from A to B and back to A causes problems in the previous rules. The Infinite Parses Test also catches such grammatical errors as the following:

S ⟶ A B C;
A ⟶ B a;
B ⟶ b A;
C ⟶ c;

In this grammar, no sentences can be generated because of the rules that have A and B as left sides (mothers). There is no termination of that path. This test reports the following types of errors:

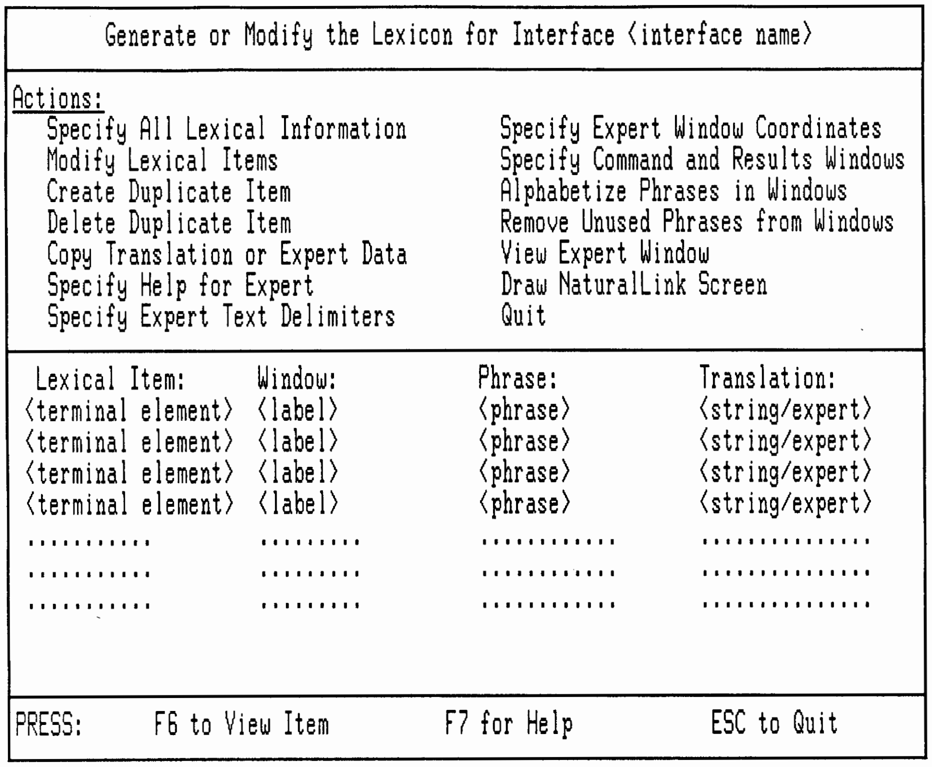
- Infinitely looping paths
- Rules that will not produce sentences

# GENERATING THE NATURALLINK LEXICON 8

## Introduction

**8.1** After you have tested the grammar and specified a screen file for the current interface, an additional option—Generate or Modify the Lexicon for the Interface—becomes available in the Create and Test Interface < interface name> screen. This option is referred to as the Lexicon Builder. When you select it, the screen shown in Figure 8-1 appears.

**Figure 8-1** **Generate or Modify Lexicon**



```
            Generate or Modify the Lexicon for Interface <interface name>

Actions:
   Specify All Lexical Information          Specify Expert Window Coordinates
   Modify Lexical Items                     Specify Command and Results Windows
   Create Duplicate Item                    Alphabetize Phrases in Windows
   Delete Duplicate Item                    Remove Unused Phrases from Windows
   Copy Translation or Expert Data          View Expert Window
   Specify Help for Expert                  Draw NaturalLink Screen
   Specify Expert Text Delimiters           Quit

 Lexical Item:      Window:           Phrase:           Translation:
<terminal element> <label>           <phrase>          <string/expert>
<terminal element> <label>           <phrase>          <string/expert>
<terminal element> <label>           <phrase>          <string/expert>
<terminal element> <label>           <phrase>          <string/expert>
...........         .........         ............      ...............
...........         .........         ............      ...............
...........         .........         ............      ...............

PRESS:      F6 to View Item          F7 for Help          ESC to Quit
```

This is a representative screen showing all the options in the Generate or Modify the Lexicon for Interface < interface name> screen. Your screen may differ, depending on the state of the files that make up the interface you are building. For example, if you have not yet specified the Command and Results windows for the current interface, only the following four options appear:

- Specify Command and Results Windows
- Alphabetize Phrases in Windows
- Draw NaturalLink Screen
- Quit

The lexicon is the semantic glue of the NaturalLink system. This file contains all the information necessary to connect each terminal element in the grammar to a natural language phrase, a screen location, and a translation in the target language. It is built interactively with the aid of a series of prompts supplied by the Generate or Modify the Lexicon for the Interface option.

Using the sequential file that contains the grammar, the Lexicon Builder determines which elements in the grammar are terminal elements (that is, elements that cannot be expanded further). For instance, in the following grammar,

```
A → B C (D)
C → D B (B)
B → E (F) G
E → e
F → d {f h}
G → g
D → d
```

'e', 'd', 'f', 'g', and 'h' are all terminal elements because none occur on the left side of a rule. Thus, they cannot be further expanded in the derivation of a sentence from this grammar.

The Lexicon Builder needs the following information for each terminal element in the grammar:

- The label of the window in the interface screen where the natural language word or phrase for this terminal element is to appear.
- The natural language word or phrase that represents the terminal element.
- The template or string the Translator will use to build the target string. You must indicate whether you will specify the translation in the target language or whether the user will provide a specific value (called an expert) at run time. (The translation field is not a required field in every case. If you do not specify a template or string in the translation field for a particular terminal element, that element's translation will be a null string.)
    - If you will specify the target language translation for the terminal element, the Lexicon Builder prompts you for the desired translation.
    - If the translation will be an expert, the Lexicon Builder prompts you for certain expert item parameters. (Experts are discussed in Chapter 9, Application Control of User Options.)

The lexicon is stored on disk in two separate files, the lexicon file (< INTFILE > .NL$) and the expert information file (< INTFILE > .NP$), where < INTFILE > is the interface file name without the extension. Do not attempt to edit these files; changes should be made only through the Lexicon Builder. To obtain a listing of the lexicon, select the List the Lexicon for the Interface option from the Create and Test Interface < interface name > screen. You can then view or print the output file you specify.

As you develop an interface, you may wish to change your interface screen after specifying many, if not all, of the lexical items. While changing the screen, you can change a window label. The Lexicon associates lexical items with windows in the screen by saving a window label, item text, and translation information for each item. The first thing the Lexicon Builder does before displaying its main screen is load any *previously specified* lexical information from the interface's lexicon file. If the Lexicon Builder finds a window label that is no longer in the screen, you must enter the window and phrase for all lexical items that were associated with that window label.

## Getting In and Out of the Lexicon Builder

**8.2** You can quit the interactive lexicon-building procedure at any time by choosing the Quit option. You can also quit from the Actions window by pressing the **ESC** key. You can save the lexical information already specified and continue building the lexicon at a later time. You should do this each time you specify a new lexicon or make substantial changes to an existing lexicon. This reduces the possibility of losing lexical information due to memory constraints of the software. (The Quit and Save procedures are explained in paragraph 8.17, Quitting the Lexicon Builder.)

When you return to the Create and Test Interface < interface name > screen (shown in Figure 7-1), the additional option List the Lexicon for the Interface is visible. This option enables you to list the current state of your lexicon to an output file. Refer to Chapter 14, Other Interface Builder Options, for a description of the List the Lexicon option.

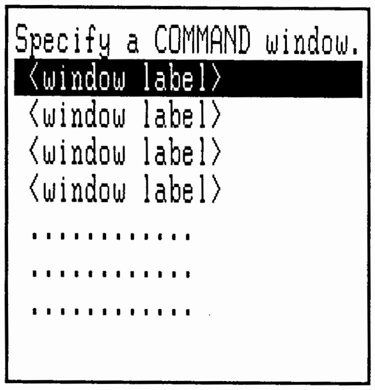
**CAUTION: Before the NaturalLink software updates the lexicon, it backs up the previous lexicon to the following files < INTFILE > .BL$ and < INTFILE > .BP$. The updated, or current, lexicon is saved to the files < INTFILE > .NL$ and < INTFILE > .NP$. If the current lexicon files are damaged or destroyed, you can copy the backup files, rename those copies using .NL$ and .NP$ filename extensions, and recreate your lexicon. When making your own backup copies of the NaturalLink interface files, however, you must be careful to copy the *current* lexicon (the two files with the .NL$ and .NP$ extensions) instead of the *old* version.**

## Specify Command and Results Windows

**8.3** The Lexicon Builder requires you to identify the Command window (the window in the NLmenu screen that contains the Sessioner commands available to the user) and the Results window (the window that shows the command sentence as the user builds it). You must specify the Command and Results windows before you can enter the window, phrase, and translation information for each terminal element in the grammar. (For more information about the Command and Results windows, see Chapter 5, Creating an NLmenu Screen.) You need to specify the Command and Results windows only once for each interface.

The Lexicon Builder prompts you for the names of the windows after you select the Specify Command and Results Windows option. First the pop-up window shown in Figure 8-2 appears, prompting for the specification of a Command window.

**Figure 8-2** **Specify Command Window**



```
Specify a COMMAND window.
<window label>
<window label>
<window label>
<window label>
............
............
............
```

This pop-up window lists labels of windows in the screen description file that you identified during the Record a New Interface's Information procedure. The only labels displayed are those for list windows and user-defined windows to which no lexical items are assigned.

Select the window you wish to assign as the Command window and press the **ENTER** key. Remember that you cannot assign lexical items to a Command window.

After you have specified the Command window, a second pop-up window prompts you to specify a Results window. Labels for list, text, display, and user-defined windows are displayed; the Command window label is not displayed. Select the window you wish to assign as the Results window. Remember that you cannot assign lexical items to a Results window.

Once you have specified the Command and Results windows, additional options appear in the Generate or Modify the Lexicon for the Interface < interface name > screen. The options that appear on this screen vary according to the state of your lexicon. For the purpose of the following discussion, assume that all options in the Generate or Modify the Lexicon screen depicted in Figure 8-1 are available.

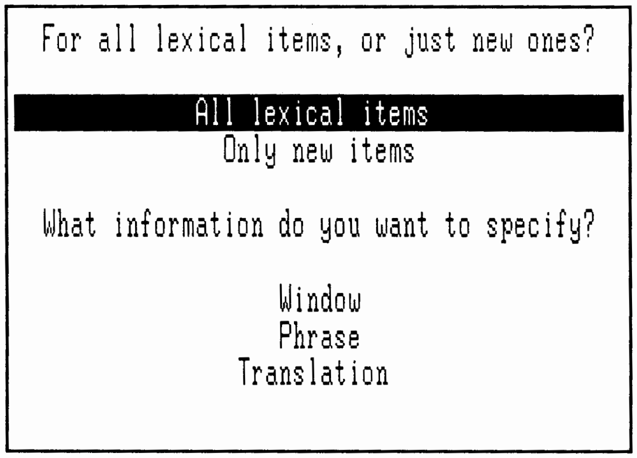
## Specify All Lexical Information

**8.4** This option enables you to specify all the information the lexicon needs for each terminal element in the grammar. This information includes:

- The words or phrases the user can select to build a command sentence
- The windows (referred to as parse windows) in which those words or phrases appear
- The translation returned to the application program when the user executes the completed command sentence

When you select the Specify All Lexical Information option, the pop-up window shown in Figure 8-3 appears, prompting for specific lexical information.

**Figure 8-3** **Specify Information Pop-Up Window**


```
For all lexical items, or just new ones?

           All lexical items
            Only new items

What information do you want to specify?

                Window
                Phrase
              Translation
```

If you have not previously specified any items for the current interface, you will normally select the All Lexical Items option. If you have either specified some items for the current interface or added new terminal elements to the grammar, you should select the Only New Items option. (If you select All Lexical Items after you have already specified some items for the lexicon, you will be prompted for information for previously specified items, as well as new ones.) The option you select is highlighted, and the cursor moves to the Window option in the lower half of the pop-up window. Select the Window, Phrase, and/or Translation option; you can select one, two, or all three. Use the arrow keys to move among items, pressing **ENTER** to select an item. Press **ENTER** again if you decide against an item already selected. The **F10** key commits the selections you have made.

For the purpose of illustration, assume that your grammar file and screen description contain lexical items and window labels similar to the following list.

| ***Lexical Item*** | ***Window Labels*** |
|---|---|
| attr_and | ACTIONS |
| equal_to | NOUNS |
| find_rel | CONNECTORS |
| for | COMPARISONS |
| numeric_expert | FEATURES |
| where_workers | QUALIFIERS |
| whose_worker_date | ATTRIBUTES |
| whose_worker_name | |
| worker_date_hired_expert | |
| worker_employee_name_expert | |

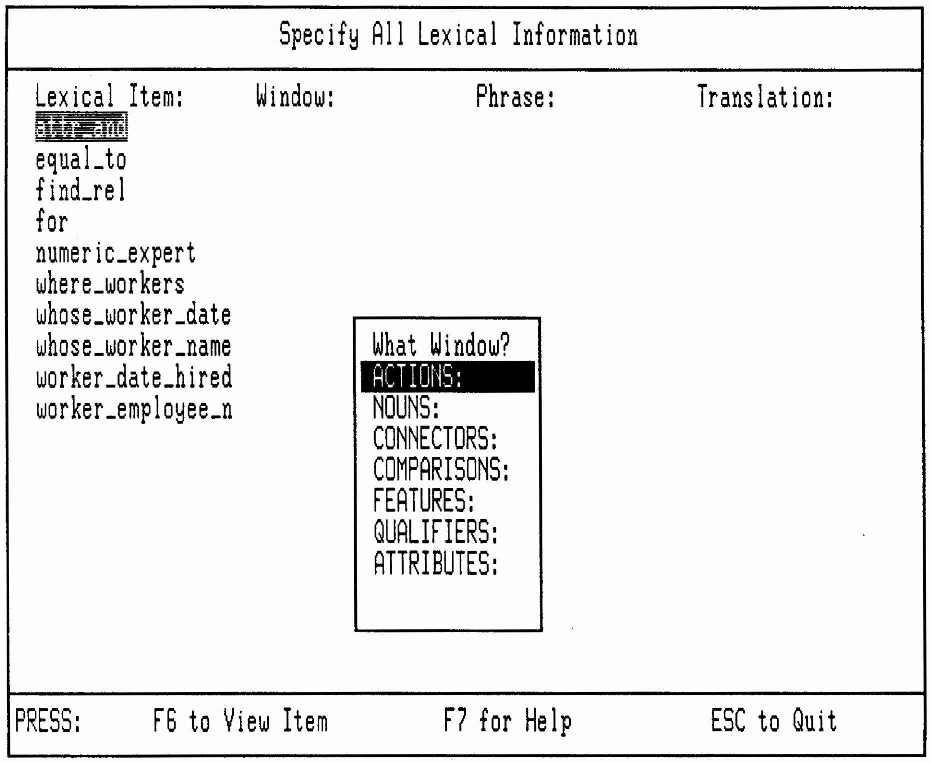
Further assume that you select the All Lexical Items option and the options Window, Phrase, and Translation in the Specify Information Pop-up Window (Figure 8-3). When you accept these selections, the screen shown in Figure 8-4 appears.

---

**NOTE:** In the following illustrations, lexical items, window labels, phrases, and translation strings or experts are added where appropriate to illustrate how the actual screens and windows described may look when you are conducting an interactive session with the Lexicon Builder. The illustrations do not match exactly the NaturalLink windows and screens displayed on your monitor. In this manual, the window in which these lexical items, window labels, and phrases are displayed is referred to as the Lexical Items window to distinguish it from the Actions window, which contains the options you select to build your lexicon.

---

**Figure 8-4** **Specify All Lexical Information**



```
                    Specify All Lexical Information

Lexical Item:      Window:          Phrase:          Translation:
attr_and
equal_to
find_rel
for
numeric_expert
where_workers
whose_worker_date
whose_worker_name       What Window?
worker_date_hired       ACTIONS:
worker_employee_n       NOUNS:
                        CONNECTORS:
                        COMPARISONS:
                        FEATURES:
                        QUALIFIERS:
                        ATTRIBUTES:

PRESS:     F6 to View Item       F7 for Help        ESC to Quit
```

The first lexical item is highlighted, and the cursor is on the first label in the What Window? pop-up window. As you make selections, the NaturalLink software guides you by highlighting items and displaying pop-up windows that prompt you for the necessary lexical information.

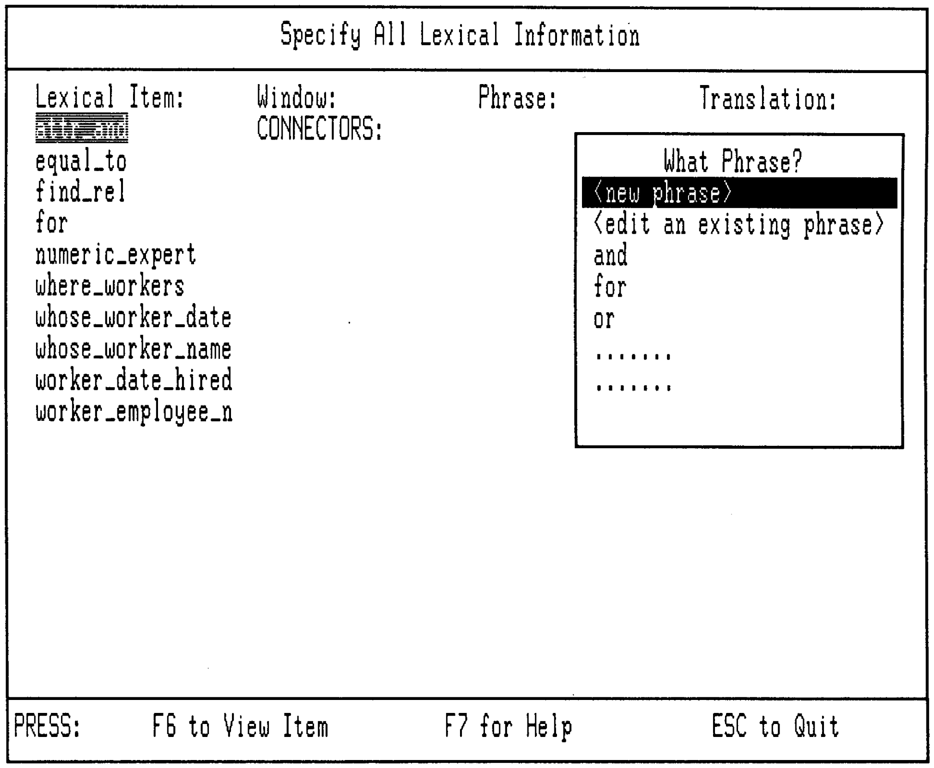
## Specify Window Labels

**8.4.1** Each terminal element in the grammar must be associated with a window in the interface screen. The What Window? pop-up window shown in Figure 8-4 contains the window labels of all list windows and user-defined windows designated in the screen description for the current interface. (The list excludes the Command and Results windows, which cannot have lexical items associated with them.) Select the label of the parse window you wish to associate with the highlighted lexical item. The Window column in the screen is updated with your selection (in this case, CONNECTORS).

## Specify a Phrase

**8.4.2** After you have specified the window label, a new pop-up window appears, prompting you to specify a word or phrase to be associated with the highlighted lexical item (Figure 8-5).

**Figure 8-5** **Phrase Selection Pop-Up Window**



```
                    Specify All Lexical Information

Lexical Item:     Window:          Phrase:          Translation:
attr_and          CONNECTORS:
equal_to                                  What Phrase?
find_rel                          <new phrase>
for                               <edit an existing phrase>
numeric_expert                    and
where_workers                     for
whose_worker_date                 or
whose_worker_name                 .......
worker_date_hired                 .......
worker_employee_n

PRESS:      F6 to View Item        F7 for Help         ESC to Quit
```

Each terminal element in the grammar to which a window has been assigned must also be associated with a natural language word or phrase in the NLmenu screen. The Lexicon Builder elicits phrase information by displaying the What Phrase? pop-up window (Figure 8-5). This window contains a list of all items that are currently in the CONNECTORS window. These include existing words and phrases, such as And, For, and Or, that you specified when you originally designed the NLmenu screen (using the Screen Builder utility) or during a previous lexicon-building session. Two additional items, < new phrase > and < edit an existing phrase >, are also listed. You have three choices:

- Enter a new phrase for the current lexical item.
- Edit an existing phrase and associate the current lexical item with the edited phrase. All lexical items associated with the original phrase will then be associated with the new phrase.
- Associate the current lexical item with any existing phrase in the parse window.

If you choose to enter a new phrase, select < new phrase > from the list of options. A pop-up window appears soliciting the new phrase for the lexical item.

If you choose to edit an existing phrase in the list, select < edit an existing phrase > from the list of options. Then select the existing phrase in the What Phrase? window that you wish to edit. The phrase is then displayed in a pop-up window so it may be edited. When you modify an existing phrase, all other lexical items associated with that phrase are then associated with the modified phrase.

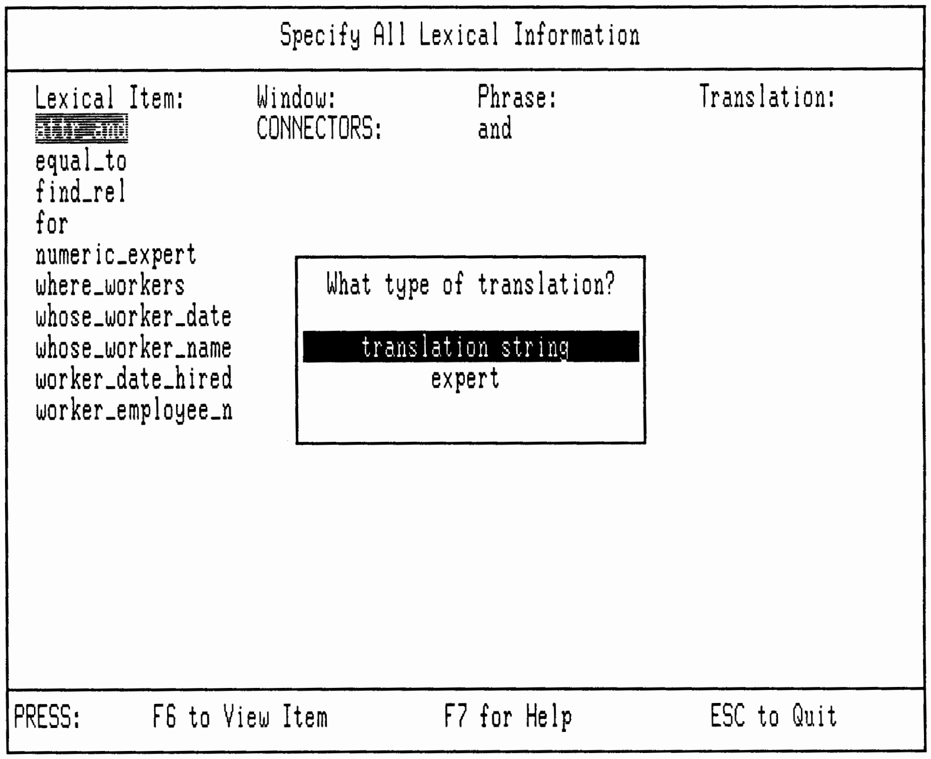
If you wish to associate the current lexical item with an existing phrase in the parse window, select the desired phrase.

After you have specified (or modified) the phrase information for the current lexical item, the Phrase column in the screen is updated. The NaturalLink software truncates phrase information that does not fit within the phrase label boundaries. The full text is saved; to view it, press the View Item/**F6** key.

**Specify a Translation**

**8.4.3** When you have completed the phrase information, a new pop-up window appears, prompting you to choose the type of translation information you wish to specify (Figure 8-6).

**Figure 8-6** **Translation Type**



```
                 Specify All Lexical Information

Lexical Item:      Window:         Phrase:         Translation:
attr_and           CONNECTORS:     and
equal_to
find_rel
for
numeric_expert
where_workers         What type of translation?
whose_worker_date
whose_worker_name         translation string
worker_date_hired               expert
worker_employee_n

PRESS:      F6 to View Item         F7 for Help         ESC to Quit
```

Each terminal element in the grammar must be associated with a translation. There are two types of translation:

- A target language string that you have specified (or a null string if you do not enter a target language string)
- An expert value that the user supplies in the course of building a NaturalLink command (experts are discussed in paragraph 8.5, Experts)

The translation string or the expert entry is returned to the application program when the user executes a command sentence. If you select Translation String, a pop-up edit window appears, prompting you to enter the target language translation string.

In some cases, the application program may require that a control character be entered as part of the translation string. If this control character is the **ENTER** key or one of the function keys, it must be entered as a hexadecimal code between two **CTRL-A** delimiters. If you do not use this special control-code sequence, Window Manager interprets the control character as an end-of-input character and does not include it in the translation field. Use the following control-code sequence to specify a translation string that includes a control character:

< **CTRL-A** > < *character code* > < **CTRL-A** >

Each **CTRL-A** appears as a happy face ☺ character. For example, for the translation string

ABC< carriage return > 123< carriage return >

you would type the following into the translation string text field:

```
ABC<CTRL-A>0D<CTRL-A>123<CTRL-A>0D<CTRL-A>
```

This control code sequence would appear on the monitor as

```
ABC[☺]0D[☺]123[☺]0D[☺]
```

Only one character code can be included within the **CTRL-A** pair. Thus, for the translation string

ABC< carriage return > < carriage return > 123

you would enter the following into the translation string field:

```
ABC<CTRL-A>0D<CTRL-A><CTRL-A>0D<CTRL-A>123
```

This would appear on the monitor as

```
ABC[☺]0D[☺][☺]0D[☺]123
```

You need to use the paired control-code sequence only for those keys that Window Manager intercepts. These include the **ENTER**, arrow, function, and editing keys. (For more information about the keys Window Manager intercepts, refer to Chapter 10, Window Manager Input Devices, in the *NaturalLink Window Manager Reference Manual.*)

**CTRL-A** by itself (that is, unpaired) is not interpreted as a delimiter and can be typed into the translation field. Other control-code characters can also be typed directly into the translation string field. Thus, if the application program requires the translation string

< **CTRL-A** > ABC< **CTRL-B** > 123

enter

```
<CTRL-A>ABC<CTRL-B>123
```

Stated differently, **CTRL-A** can be typed directly into a translation string field if it is not followed by another CTRL-A. However, if the translation string is

< **CTRL-A** > < carriage return > < **CTRL-A** >

you need to delimit the first **CTRL-A** by typing a pair of **CTRL-As** with the hexadecimal code for **CTRL-A** (01) between them, another pair of **CTRL-As** with the hexadecimal code for carriage return (0D) between them, and finally the last **CTRL-A**. The last **CTRL-A** is not paired, since you delimited the first **CTRL-A**, so you can type it directly into the translation field. This is what you would type for the sequence:

```
<CTRL-A>01<CTRL-A><CTRL-A>0D<CTRL-A><CTRL-A>
```

On the monitor, this would be displayed as

```
[☺]01[☺][☺]0D[☺][☺]
```

When you display translation strings containing control code/control character configurations, be aware of the following peculiarities:

- If you select the View a File option to view a Translation Test output file, that output file is displayed in a Window Manager file window. File windows start a new line of display when Window Manager encounters any of the following character codes or character code pairs:

  0D, 0A, 0D0A, 0A0D

- If you display a translation containing a carriage return through the NaturalLink Sessioner demonstration program, the carriage return is displayed as a musical note character ♪. If the window is not a file window, Window Manager does not interpret the carriage return to mean begin the next line.

- The MS-DOS TYPE command interprets control characters in the strictest sense. This means that the carriage return character, 0D, alone, causes the cursor to return to the beginning of the current line, not the beginning of a new line.

- Do not use the character code 00 in a translation string. The NaturalLink software ignores it, and you will lose the rest of your translation. In addition, the vertical bar character (|) and the exclamation point (!) are reserved characters. They are mapped to hexadecimal values 1E (▲) and 1F (▼), respectively. These special characters will be visible in your lexicon file.

## Experts

**8.5** Experts require the user to enter specific values for particular terminal elements at run time. The user enters this information into an expert pop-up (except in the case of user-defined experts). When displaying the expert pop-up window, the NaturalLink software prompts the user with a phrase like the following: "Enter a specific number in the inclusive range 50-100" or "Enter a specific date." The phrase specified for the lexical item appears in the expert pop-up window. (In these examples, "specific number" and "specific date" are the phrases specified for the lexical items.) NaturalLink expert routines perform validation checks on the specific values that are entered and pass these values to the Translator for inclusion in the target language string.

You select the type of expert to present to the user. NaturalLink provides five types of experts:

- Alphanumeric — Accepts any alphanumeric value. Length must be specified; range parameters are optional.

- Numeric — Accepts any numeric value. Length must be specified; range parameters are optional.

- Date — Accepts date values only. Format must be specified; range parameters are optional.

- List — Enables the user to make a selection from a list of possible values, for example, a list of countries or major cities. List experts reduce the possibility of error by limiting the user's options. Typing errors are also reduced. When building a list expert, you must specify the number of items that will be in the list, then specify the list.

- User-Defined — Controlled by the application program. You must write the necessary expert handling routines and specify an identifying code for each of them in the NaturalLink user software. (Refer to Chapter 9, Application Control of User Options, for more information.)

When you select Expert as the type of translation you wish to specify, a pop-up window appears, displaying a list of the five expert types you can select.

If you select alphanumeric or numeric, you are prompted for the maximum length of the expert. Alphanumeric experts cannot exceed 76 characters (the maximum length of a one-line field with one border character and one space on each side of the expert). Numeric experts cannot exceed 50 digits. Maximum length is not applicable for date, list, or user-defined experts.

Next, a prompt asks if you want to specify a range. (Date experts also require an answer to the Range? prompt.)

### Alphanumeric Expert

**8.5.1** If you select Yes for the range option for alphanumeric expert, a pop-up window prompts for the minimum and maximum range specifications.

After specifying minimum and maximum range values for alphanumeric experts, you must also indicate whether those values are, themselves, to be valid entries. If they are, select Yes for the Inclusive? prompt that appears at the bottom of the range specification pop-up window; otherwise, select No.

### Numeric Expert

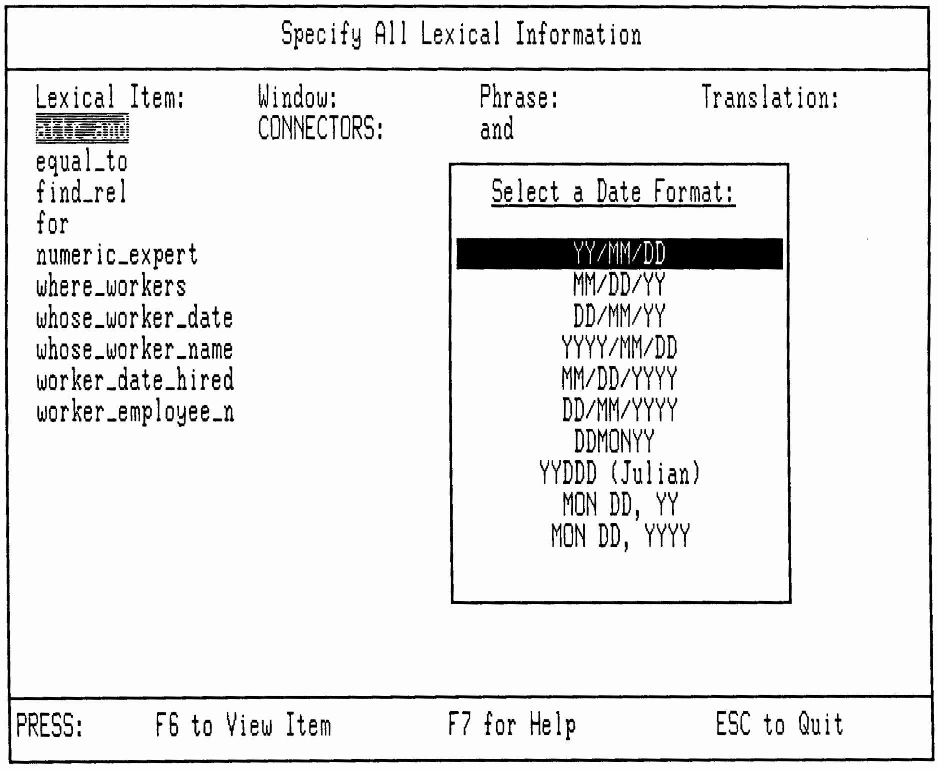
**8.5.2** The same general set of alphanumeric expert prompts appears if you select the numeric expert option, but the prompting pop-up windows are slightly different. With numeric experts, you have the additional option of specifying an increment for a range. This provides greater control over user entries. For example, if you specify minimum and maximum inclusive range values of 10 and 100, respectively, and an increment of 5, the expert routine accepts only user entries of 10, 15, 20, and so forth.

The NaturalLink software does not support increments if the numeric values consist of more than 9 digits. If you specify 10 digits or more as the maximum length of the numeric expert, you are not prompted for an increment value.

### Date Expert

**8.5.3** When you select the date expert option, you first select a date format from the pop-up window shown in Figure 8-7.

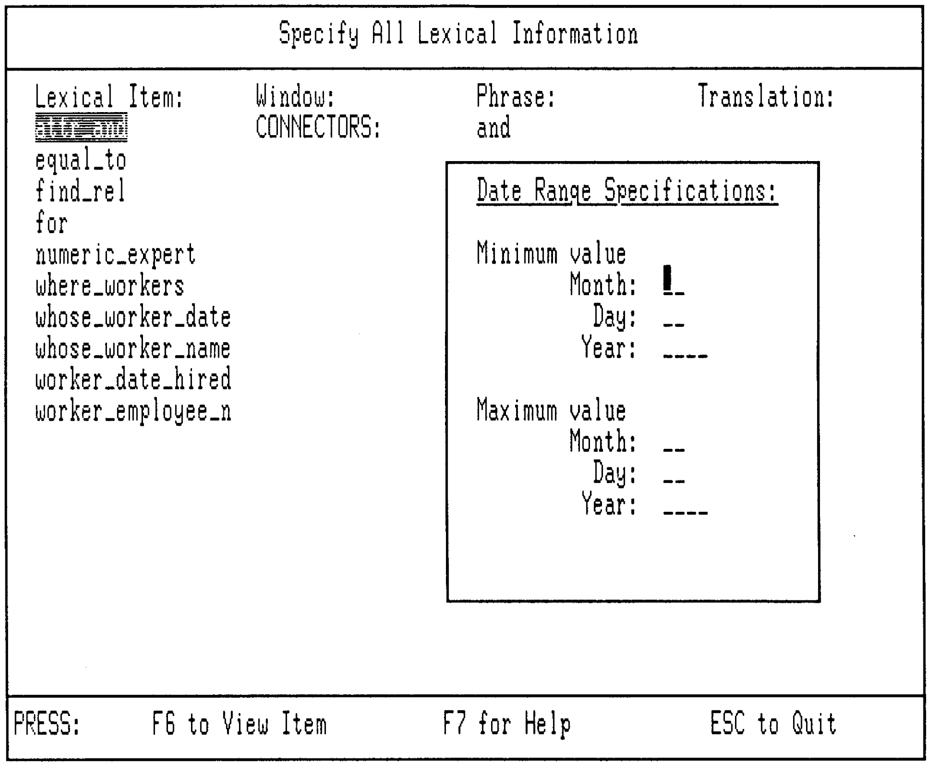
**Figure 8-7** **Date Format**



```
                     Specify All Lexical Information

Lexical Item:      Window:          Phrase:          Translation:
attr_and           CONNECTORS:      and
equal_to
find_rel                          Select a Date Format:
for
numeric_expert                          YY/MM/DD
where_workers                           MM/DD/YY
whose_worker_date                       DD/MM/YY
whose_worker_name                       YYYY/MM/DD
worker_date_hired                       MM/DD/YYYY
worker_employee_n                       DD/MM/YYYY
                                        DDMONYY
                                     YYDDD (Julian)
                                       MON DD, YY
                                      MON DD, YYYY

PRESS:      F6 to View Item        F7 for Help          ESC to Quit
```

Next, you must respond Yes or No to a range specification prompt similar to that for alphanumeric and numeric experts. Date range values are specified in the pop-up window shown in Figure 8-8, which also prompts you to specify whether the range values are inclusive.

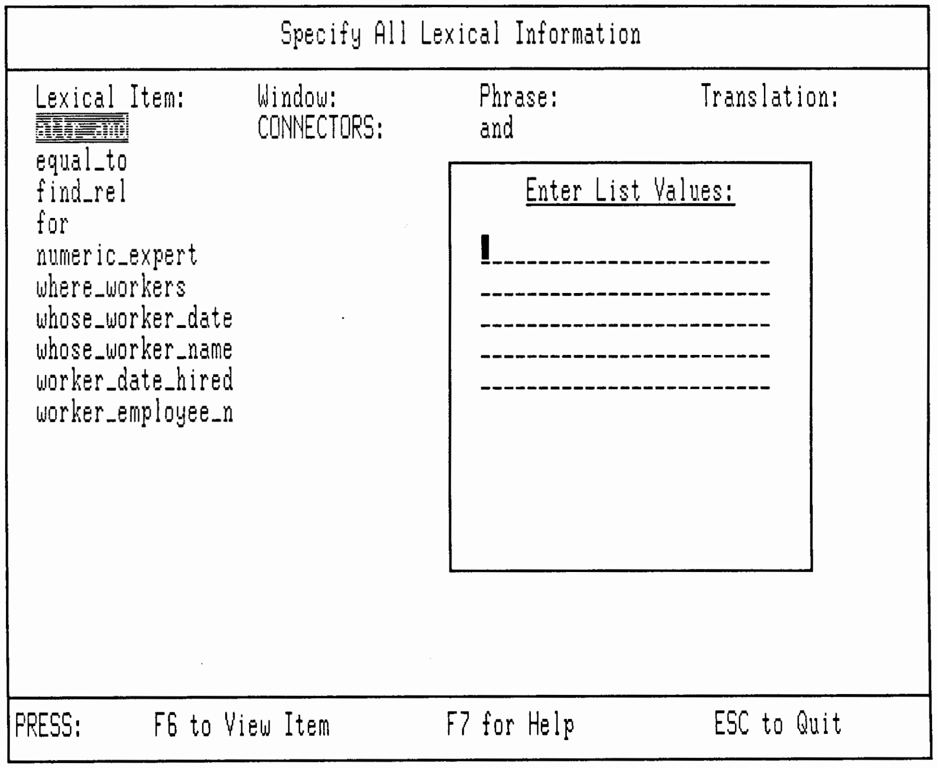
**Figure 8-8** **Date Range Specifications**



```
                    Specify All Lexical Information

Lexical Item:      Window:          Phrase:          Translation:
[illegible]        CONNECTORS:      and
equal_to
find_rel                            Date Range Specifications:
for
numeric_expert                      Minimum value
where_workers                               Month: _
whose_worker_date                             Day: __
whose_worker_name                            Year: ____
worker_date_hired
worker_employee_n                   Maximum value
                                            Month: __
                                              Day: __
                                             Year: ____

PRESS:     F6 to View Item        F7 for Help          ESC to Quit
```

**List Expert** **8.5.4** After selecting the list expert option, you specify the number of items to be displayed in the list. A pop-up window similar to the one shown in Figure 8-9 prompts for the specific values that will appear in the interface screen.

**Figure 8-9** **Specify Values for List Expert**



```
                      Specify All Lexical Information

 Lexical Item:      Window:          Phrase:          Translation:
 all_and            CONNECTORS:      and
 equal_to
 find_rel                              Enter List Values:
 for
 numeric_expert                      _________________________
 where_workers                       _________________________
 whose_worker_date                   _________________________
 whose_worker_name                   _________________________
 worker_date_hired                   _________________________
 worker_employee_n

PRESS:      F6 to View Item        F7 for Help        ESC to Quit
```

---

**NOTE:** The NaturalLink software stores information about experts in the file < INTFILE.NP$ > . If you attempt to modify an existing lexicon and the expert file does not exist, all experts specified in the lexicon default to alphanumeric experts when you enter the Generate or Modify the Lexicon utility.

---

## User-Defined Expert

**8.5.5** If you select the user-defined expert option, a prompt asks for the two-digit code that identifies the expert to the Sessioner. Enter only numeric values for expert codes. Entering any nonnumeric character causes the code to default to 0 (zero). (For more information about writing user-defined expert routines, refer to Chapter 9, Application Control of User Options.)

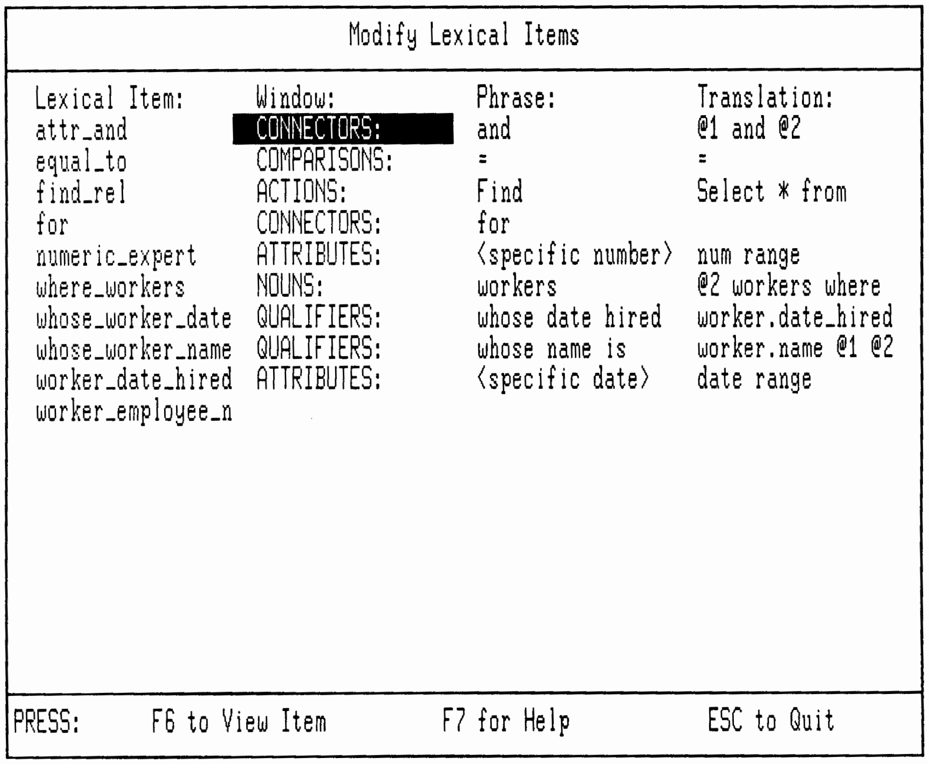
**CAUTION: If you plan to use more than one interface in a given application, the codes that identify user-defined experts must be unique in respect to both the application they are associated with and all other interfaces in the application. A fatal error may result if the same code identifies more than one user-defined expert in an application.**

## Modify Lexical Items

**8.6** The Modify Lexical Items option enables you to change any of the entries you specified during a lexicon-building session. The screen shown in Figure 8-10 appears when you select this option.

**NOTE:** In the following examples, additional window, phrase, and translation items have been added for illustration purposes.

**Figure 8-10** **Modify Lexical Items**



```
                          Modify Lexical Items

 Lexical Item:      Window:            Phrase:            Translation:
 attr_and           CONNECTORS:        and                @1 and @2
 equal_to           COMPARISONS:       =                  =
 find_rel           ACTIONS:           Find               Select * from
 for                CONNECTORS:        for
 numeric_expert     ATTRIBUTES:        <specific number>  num range
 where_workers      NOUNS:             workers            @2 workers where
 whose_worker_date  QUALIFIERS:        whose date hired   worker.date_hired
 whose_worker_name  QUALIFIERS:        whose name is      worker.name @1 @2
 worker_date_hired  ATTRIBUTES:        <specific date>    date range
 worker_employee_n

PRESS:      F6 to View Item          F7 for Help          ESC to Quit
```

Each terminal element in the grammar is listed under the Lexical Item heading. The other three headings list the window, phrase, and translation information specified during the lexicon-building process. The cursor is initially on the first label under the Window heading.

Select the item you wish to modify. The same series of pop-up windows that appeared when you initially specified the item is displayed again. You perform essentially the same steps to modify the lexical information as you did to specify that information initially. The cursor remains in the Lexical Items window until you press the **ESC** key or the **F10** key to end the Modify Lexical Items session. (To view the specified lexical information for a particular item without the truncation that occurs in the Lexical Items window, place the cursor on that item and press the **F6** key.)

If you have already specified a window label and a phrase for some lexical item and decide to change the window label, the old, unused phrase remains in the original window. Similarly, if you have already specified a phrase for some lexical item and decide to change that phrase, the old, unused phrase remains in the window. For information about eliminating unused phrases, see paragraph 8.14, Remove Unused Phrases From Windows.

## Create Duplicate Item

**8.7** This option enables you to duplicate all items in a lexical entry except its phrase. This is useful when multiple phrases map to the same terminal element. Selecting this option causes the cursor to be placed in the Lexical Items window. You can then select the item(s) for which you wish to create duplicate lexical items.

The Create Duplicate Item option is highlighted, and the cursor is on the first lexical item in the list. Select the item you wish to duplicate. The lexical item, window, and translation of the item you selected are duplicated and inserted directly below the current lexical item. The phrase for the duplicated item is left blank. You should select the Modify Lexical Items option to enter new phrase information. You can create several duplicate items in a session. The cursor remains in the Lexical Items window until you press the **ESC** key or the **F10** key.

**CAUTION: Because of the way the NaturalLink software allocates memory, you should quit the Generate or Modify the Lexicon operation and save the changes made to your lexicon after creating no more than 8 to 10 duplicate items. Then exit the Create and Test Interface screen and return to the main Interface Builder screen, at which time memory is cleared. This is a precautionary measure to protect your lexicon file. When you enter the Generate or Modify the Lexicon screen again, memory is reallocated.**

## Delete Duplicate Item

**8.8** This option enables you to delete a duplicate item from the lexicon. You can delete only those items created with the Create Duplicate Item option. The cursor remains in the Lexical Items window until you press the **ESC** key or the **F10** key to return to the Actions window or until you have deleted all duplicate entries.

The phrase associated with the deleted duplicate item remains in the window associated with that item but is no longer used by the Sessioner. See paragraph 8.14, Remove Unused Phrases From Windows, for information about removing unused phrases.

## Copy Translation or Expert Data

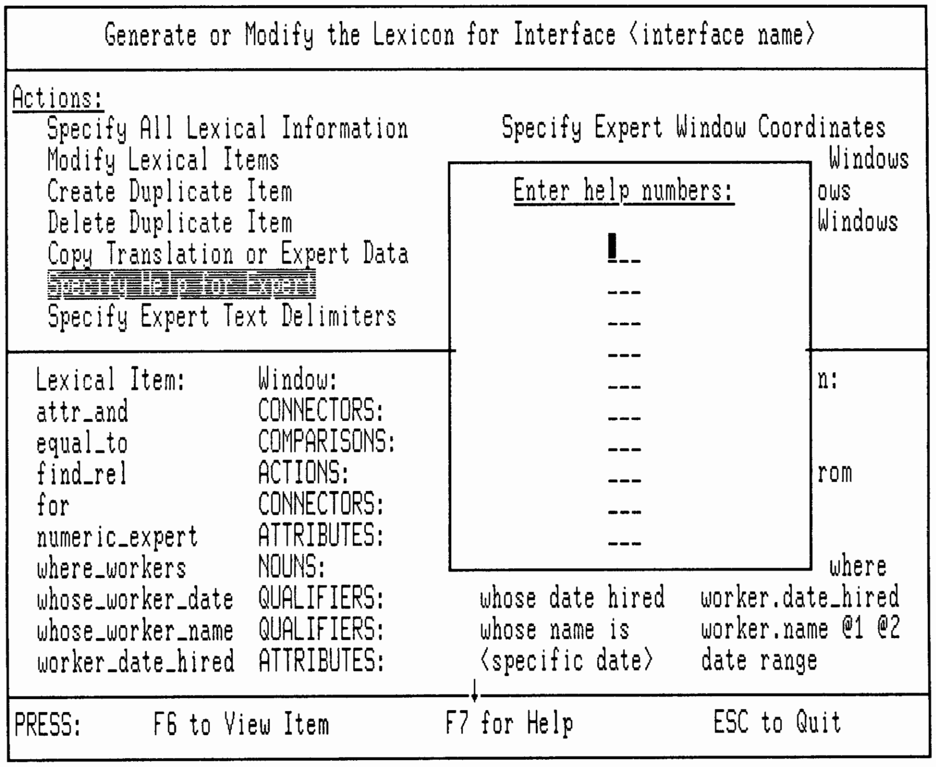
**8.9** This option enables you to copy translation or expert information from one lexical item to another. When you select Copy Translation or Expert Data, the cursor is placed on the first translation or expert in the list. Select the translation or expert data you want to copy (the source item). That translation information and its associated lexical item are then highlighted. Next, select the translation item to which you want to copy (the destination item). The source translation string or expert is copied to the destination item. The cursor remains in the Lexical Items window until you press the **ESC** key or the **F10** key to return to the Actions window.

## Specify Help for Expert

**8.10** This option enables you to attach Help messages to the experts presented to the user in the interface screen. These Help messages are created with Screen Builder or Message Builder and must be in the Help message file attached to the interface screen. (Help messages for user-defined experts must be specified in the application program.) When you select this option, the cursor is placed in the Lexical Items window. You can then choose an expert for which you wish to specify help.

The Specify Help for Expert option is highlighted, and the cursor is on the first of the available experts in the Translation list. Select the alphanumeric, numeric, date, or list expert for which you wish to specify one or more Help messages. The screen shown in Figure 8-11 appears.

**Figure 8-11** **Specify Help Numbers**



```
          Generate or Modify the Lexicon for Interface <interface name>

Actions:
   Specify All Lexical Information          Specify Expert Window Coordinates
   Modify Lexical Items                 ____________________________ Windows
   Create Duplicate Item               |   Enter help numbers:      |ows
   Delete Duplicate Item               |                            |Windows
   Copy Translation or Expert Data     |          _                 |
   Specify Help for Expert             |          ---               |
   Specify Expert Text Delimiters      |          ---               |
                                       |          ---               |
 Lexical Item:      Window:            |          ---               |n:
 attr_and           CONNECTORS:        |          ---               |
 equal_to           COMPARISONS:       |          ---               |
 find_rel           ACTIONS:           |          ---               |rom
 for                CONNECTORS:        |          ---               |
 numeric_expert     ATTRIBUTES:        |          ---               |
 where_workers      NOUNS:             |____________________________| where
 whose_worker_date  QUALIFIERS:          whose date hired   worker.date_hired
 whose_worker_name  QUALIFIERS:          whose name is      worker.name @1 @2
 worker_date_hired  ATTRIBUTES:          <specific date>    date range

PRESS:      F6 to View Item          F7 for Help           ESC to Quit
```

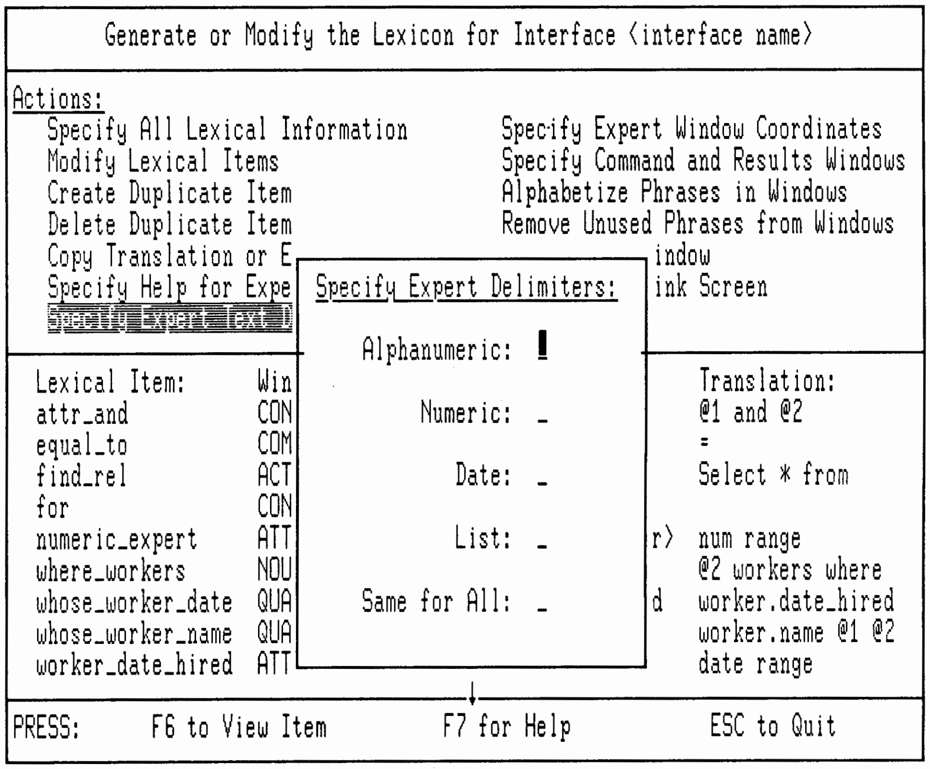
Enter the identifying numbers (up to 10 numbers) of the Help message(s) you wish to associate with the expert item. The order in which you specify Help messages is the order in which they will appear to the user. Press the **F10** key to accept the numbers you have entered, or press the **ESC** key to abort this operation without accepting any identifying numbers you have entered. Use the **DEL** key to delete numbers you no longer want. When you press **F10** or **ESC**, the cursor remains in the Lexical Items window until you press **ESC** or the **F10** key again to return to the Actions window.

Help messages cannot be tested through IBUILD. You can use the demonstration program (NLLCDEMO.C) included with the package to test these Help messages after the interface file is generated.

## Specify Expert Text Delimiters

**8.11** In some applications, expert text must be offset by specific characters (such as quotation marks or brackets). These delimiters vary according to the application program and the type of information being processed. An alphabetic string might need quotation marks, while the application might be able to interpret a numeric string with no accompanying delimiters. The NaturalLink software must have this information to add these delimiters to the expert text automatically, as required. When you select the Specify Expert Text Delimiters option, the pop-up window shown in Figure 8-12 elicits the necessary information.

**Figure 8-12** **Specify Expert Delimiters**



```
        Generate or Modify the Lexicon for Interface <interface name>

Actions:
   Specify All Lexical Information          Specify Expert Window Coordinates
   Modify Lexical Items                     Specify Command and Results Windows
   Create Duplicate Item                    Alphabetize Phrases in Windows
   Delete Duplicate Item                    Remove Unused Phrases from Windows
   Copy Translation or E                    indow
   Specify Help for Expe  Specify Expert Delimiters:  ink Screen
   Specify Expert Text D
                              Alphanumeric: ▮
 Lexical Item:       Win                                   Translation:
 attr_and            CON          Numeric: _               @1 and @2
 equal_to            COM                                   =
 find_rel            ACT             Date: _               Select * from
 for                 CON
 numeric_expert      ATT             List: _           r)  num range
 where_workers       NOU                                   @2 workers where
 whose_worker_date   QUA     Same for All: _           d   worker.date_hired
 whose_worker_name   QUA                                   worker.name @1 @2
 worker_date_hired   ATT                                   date range

PRESS:      F6 to View Item         F7 for Help          ESC to Quit
```

You can enter the specific beginning delimiter (such as quotation marks or a bracket) that will be associated with an alphanumeric string, a numeric value, a date, or a list value. (User-defined experts are not given delimiters. If delimiters are required, they can be returned with the expert value for the user-defined expert.) The closing delimiter is automatically associated with the following delimiters:

(, [, <, or {

If the same pair of delimiters is to be associated with all types of expert values, you can enter a value in the Same for All category. Press the **F10** key to accept the delimiter(s) you have specified and return to the Actions window, or press **ESC** to abort the operation. Use the **DEL** key to erase options.

## Specify Expert Window Coordinates

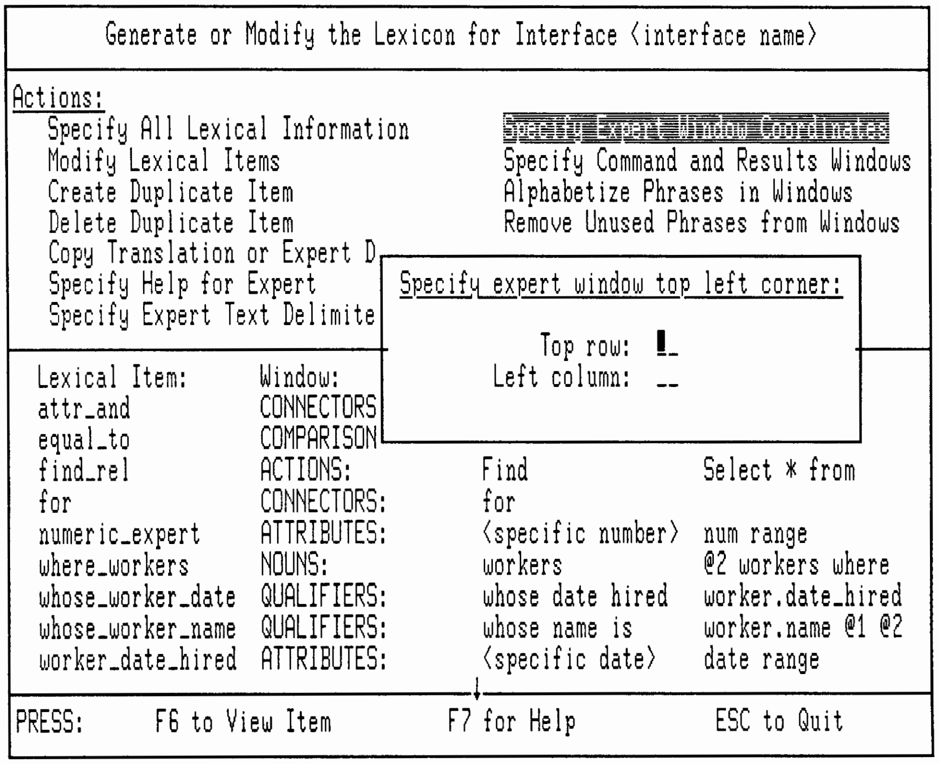
**8.12** This option enables you to specify the pop-up window position for alphanumeric, numeric, date, and list experts. (Window coordinates for user-defined experts must be specified in the application program.) When you select this option, the cursor is placed in the Lexical Items window. You can then select the expert for which you wish to specify coordinates.

The Specify Expert Window Coordinates option is highlighted, and the cursor is on the first appropriate expert in the Translation column. Select the expert for which you wish to specify window coordinates.

### Alphanumeric, Numeric, and Date Experts

**8.12.1** If you select an alphanumeric, numeric, or date expert, the screen shown in Figure 8-13 is displayed. Alphanumeric and numeric experts always use five rows of space; date experts use seven rows. The number of columns used depends on the length of the window label designated for the expert. You specify only the top left corner of the expert window.

**Figure 8-13** **Specify Expert Window Coordinates**



```
                Generate or Modify the Lexicon for Interface <interface name>

Actions:
   Specify All Lexical Information            Specify Expert Window Coordinates
   Modify Lexical Items                       Specify Command and Results Windows
   Create Duplicate Item                      Alphabetize Phrases in Windows
   Delete Duplicate Item                      Remove Unused Phrases from Windows
   Copy Translation or Expert D
   Specify Help for Expert      Specify expert window top left corner:
   Specify Expert Text Delimite
                                           Top row: _
  Lexical Item:      Window:           Left column: __
  attr_and           CONNECTORS
  equal_to           COMPARISON
  find_rel           ACTIONS:           Find                Select * from
  for                CONNECTORS:        for
  numeric_expert     ATTRIBUTES:        <specific number>   num range
  where_workers      NOUNS:             workers             @2 workers where
  whose_worker_date  QUALIFIERS:        whose date hired    worker.date_hired
  whose_worker_name  QUALIFIERS:        whose name is       worker.name @1 @2
  worker_date_hired  ATTRIBUTES:        <specific date>     date range

PRESS:     F6 to View Item          F7 for Help          ESC to Quit
```

**List Experts**

**8.12.2** This type of expert allows specification of the top left corner and, optionally, the bottom right corner window coordinates, because the number of rows used depends on the number of items designated for the list. The minimum number of rows required is four. If you do not specify the bottom right corner, the NaturalLink software will expand the window as needed to accommodate the list. The pop-up window in which you specify coordinates for list experts is similar to that for the other experts except that you will be able to specify the bottom corner coordinates.

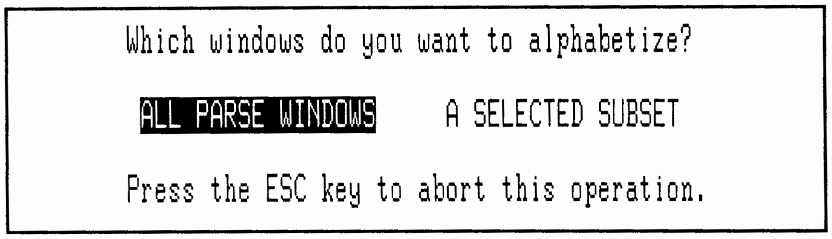
When you specify window coordinates for an expert window, the cursor remains in the Lexical Items window. You can select another expert for which to specify window coordinates, or you can return to the Actions window by pressing the **ESC** key or the **F10** key.

To delete window coordinates you have previously specified and accept the default window coordinates supplied by the NaturalLink software, use the **DEL** key.

## Alphabetize Phrases in Windows

**8.13** This option enables you to alphabetize the phrases in the parse windows of the NLmenu screen. When you select this option, the pop-up window shown in Figure 8-14 appears.

**Figure 8-14** **Alphabetize Phrases in Windows**


```
Which windows do you want to alphabetize?

ALL PARSE WINDOWS      A SELECTED SUBSET

Press the ESC key to abort this operation.
```

If you select All Parse Windows, the phrases in all parse windows will be alphabetized. If you choose A Selected Subset, a pop-up window will be shown with a list of window labels to choose from. Select the window(s) containing the phrases you wish to be alphabetized. Use the arrow keys to move among items in this pop-up window, and press **ENTER** to select an item. If you decide against an item already selected, press **ENTER** again. Selected window labels are displayed in reverse video. Note that you can alphabetize items in the Command window when you use the Selected Subset option. Press the **F10** key to execute the operation.

This procedure puts all the phrases in the specified window(s) into alphabetical order, permitting more efficient use of the Window Manager search feature in the final user interface screen. (See the *NaturalLink Window Manager Reference Manual* for information about the search feature.)

To view the alphabetized windows, select the Draw NaturalLink Screen option described later in this chapter.

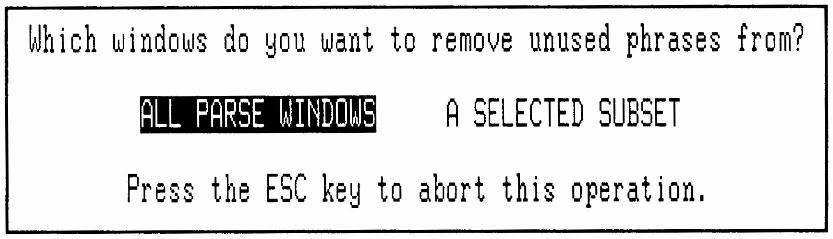
## Remove Unused Phrases From Windows

**8.14** This option enables you to remove unused phrases from the window(s) to which they were once assigned. There are four reasons why a phrase may be unused:

- You change a window associated with a lexical item. In the course of building or modifying your lexicon, you may change the window associated with a lexical item. The phrase for that lexical item then appears in two places: (1) in the new window with which the lexical item is now associated, and (2) in the old window from which the phrase's lexical item has been removed. However, the phrase will not be used in the old window since it is no longer associated with a lexical item assigned to that window.
- You specify a new phrase to replace a phrase currently associated with a lexical item. Both the new phrase and the old (unused) phrase will appear in the window assigned to that lexical item.
- You use the Delete Duplicate Item option. If you use this option, the phrase associated with the deleted duplicate item remains (because you first created the duplicate item using the Create Duplicate Item option, then entered the phrase information using the Modify Lexical Items option).
- You specify a phrase that you do not associate with a lexical item. This may happen when you design the NLmenu screen with the Screen Builder utility.

The Remove Unused Phrases From Windows option enables you to delete all unused phrases from windows in your NLmenu screen. When you select this option, the pop-up window shown in Figure 8-15 appears.

Figure 8-15 **Remove Unused Phrases**



```
Which windows do you want to remove unused phrases from?

        ALL PARSE WINDOWS      A SELECTED SUBSET

          Press the ESC key to abort this operation.
```

If you choose A Selected Subset, a pop-up window appears, with a list of window labels to choose from. Select the window(s) containing the phrase(s) you wish to remove. Use the arrow keys to move among items, and press **ENTER** to select an item. Selected window labels are displayed in reverse video. Press **ENTER** again if you decide against an item already selected. Press the **F10** key to execute the operation.

## View Expert Window

**8.15** Select this option if you wish to view an expert pop-up window. (You must specify the phrase for that expert before you can view the window.) The cursor appears on the first of the available expert items in the Lexical Items window. Select the expert item whose pop-up window you wish to view; the designated pop-up window is displayed. You can enter values for the expert window, but the NaturalLink software does not perform validation checks on these values.

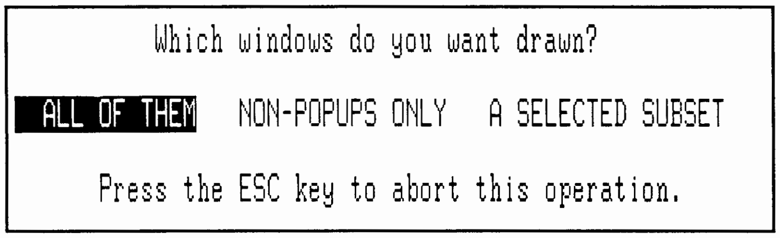
Press the **ENTER** key, the **F10** key, or the **ESC** key to return to the Lexical Items window.Then press the **F10** or the **ESC** key to return to the Actions window.

You cannot test Help messages when viewing expert windows. After the interface is generated, you can test Help messages for expert windows using the demonstration program (NLLCDEMO.C) included with the package.

## Draw NaturalLink Screen

**8.16** This option enables you to view the current NLmenu screen. When you select this option, the pop-up window shown in Figure 8-16 appears.

**Figure 8-16** **Draw NaturalLink Screen**



```
        Which windows do you want drawn?

 ALL OF THEM   NON-POPUPS ONLY   A SELECTED SUBSET

     Press the ESC key to abort this operation.
```

Select the option you want and press the **ENTER** key. If you choose A Selected Subset, an additional pop-up window appears, prompting you to select the window(s) you wish to see. Use the arrow keys to move among items, and press **ENTER** to select an item. Press **ENTER** again if you decide against an item already selected. Press the **F10** key to commit your selection. After you view the screen, press the **ENTER** key, the **F10** key, or the **ESC** key to return to the Actions window.

You cannot view Help messages when viewing the screen. You can view Help messages using Screen Builder or the demonstration program (NLLCDEMO.C) included with the package.

## Quitting the Lexicon Builder

**8.17** You can continue the lexicon-building process until you have specified all the information required by the Lexicon Builder for all terminal elements in the grammar, or you can quit at any time. During an Actions window operation, you can return to the Generate or Modify the Lexicon Interface < interface name > menu by pressing the **ESC** key (the number of times you must press **ESC** depends on what step you are performing in the operation).

Once you are in the main menu, you can select the Quit option. The Lexicon Builder then asks if you want to save the changes made to the lexicon. You can do this by selecting Yes, or you can quit without saving by selecting No. If you select Yes, the screen file, the lexicon file, and the expert file are updated.

## Function Keys

**8.18** The following list explains how function keys work during the lexicon-building procedure.

- View Item/**F6** — When the cursor is on an item in the Lexical Items window, press **F6** to display all the lexical information specified for that lexical item. The text of items viewed is not truncated but appears in full. The **F6** key is valid only when you have selected one of these options:
  - Specify All Lexical Information
  - Modify Lexical Items

- Create Duplicate Item
- Delete Duplicate Item
- Copy Translation or Expert Data
- Specify Help for Expert
- Specify Expert Window Coordinates
- View Expert Window

If the cursor is in the Actions window and you wish to view the lexical information specified for a particular item, simultaneously press the **CTRL** key and the **PgDn** key to place the cursor in the Lexical Items window. Then place the cursor on the lexical item for which you wish to view information and press the **F6** key.

- Help/**F7** — Help messages are provided for all options in the Lexicon Builder utility. Place the cursor on the option for which you wish to read the Help message(s) and press the **F7** key. If more than one Help message is associated with a particular option, a statement at the bottom of the Help screen prompts you to press the **F7** key again for more help. Press the **ENTER** key to return to the screen.

- Quit/**ESC** — The **ESC** key enables you to quit the operation you are currently performing. If you are performing one of the Actions operations, the **ESC** key returns you to the previous screen. If you are in the Generate or Modify the Lexicon for the Interface screen, the **ESC** key returns you to the Create and Test Interface < interface name > screen.

- **F10** and **ENTER** — The **F10** key accepts multiple-item selections. In general, you select options with the **ENTER** key and accept or execute those options with the **F10** key.

# APPLICATION CONTROL OF USER OPTIONS 9

## Introduction

**9.1** At times it may be necessary or desirable to control user options from the application program. You can do this through two NaturalLink options:

- User-defined experts
- Dynamic lexical items

Each option is designed to solve a specific type of problem. If both will solve your problem, use user-defined experts; that option provides a faster and more memory-efficient solution than dynamic lexical items.

This chapter explains how to use each of these methods to control user options from the application program. User-defined experts are discussed first, then dynamic lexical items. Next the chapter presents examples that demonstrate how you decide which of these two options to choose. Finally, sample calls for dynamic lexical items are listed at the end of the chapter.

## User-Defined Experts

**9.2** This section explains the uses for user-defined experts and the procedure for writing a user-defined expert routine.

### Uses for User-Defined Experts

**9.2.1** With the NaturalLink Toolkit you can tailor expert-handling routines to special needs. These experts are useful when the standard NaturalLink type-in, date, and list experts are not sufficiently flexible for certain application requirements. For example, in standard NaturalLink experts, items and parameters are specified when the lexicon is built and become a permanent part of the interface, while items in user-defined experts can be added, deleted, or changed at your discretion.

This capability is often helpful. Assume you are creating a NaturalLink interface to an online stock quote service. Users of the interface typically would ask for stock quotes on only a small list of all the available stocks. The stock list would, of course, be different for each user. Ideally, each user should be able to create his or her own unique stock list and change that list at any time. NaturalLink user-defined experts provide this capability.

When the user, in building a query, specifies the stocks for which quotes are to be obtained, the application's user-defined expert routine is called to present the user's stock list (along with an option to type in a different stock). Each time the user changes the list, it is updated and saved as part of the application. This eliminates the need for the user to memorize stock symbols for the current stocks of interest and type them in each time the query is made.

There are additional ways to change the options available to the user through user-defined experts.

- In a relational database, a user-defined expert can present the user with a list of all current values for a specific field. When the user-defined expert routine is called, it queries the database for the values in that field and displays them in a list window from which the user can choose one or more.

- In a text database, the query process might require a list of synonyms to be provided every time a specific word, such as *car*, is specified. The application can enable the user to create lists of words that are synonyms of each other. The user-defined expert elicits a word, probably through an edit window, and attempts to match the value with the lists of synonyms already specified. If the word appears in the lists, the user can be given the choice of including all or part of the previously defined synonyms in the query, instead of having to type them each time.

Like other experts, user-defined experts cannot contain slots. Therefore, user-defined experts should not occur as predicates with arguments in the semantic substitution list in any rules in the grammar. If they do, no substitution will be performed.

For specific information about using the user-defined experts option, refer to Chapter 15, The NaturalLink Sessioner, and Appendix C, High-Level Language Interface.

**Procedure** **9.2.2** The NaturalLink software enables you to write a user-defined expert routine that elicits the specific data directly from the user. When the NaturalLink Sessioner encounters any item that has been specified as a user-defined expert, it calls a routine in the application program called NLXEXP and passes it a code specified in the lexicon-building portion of the IBUILD utility, identifying which user-defined expert is being invoked. The NLXEXP routine must determine how to elicit the specific data by checking the identifying code. Then it must call its own routines to handle the expert.

The application program controls everything involved in eliciting the user data. You define all the windows or screens required with the aid of the Screen Builder. Your own expert-handling routines make the necessary Window Manager calls to display those windows and receive the information from the user. If, as in the stock quote example, the data for the expert window is to come from a file, the application routines must read the file, load the screen to be used, put the data in the window, add the window, receive the user's choice, delete the window, and unload the screen.

Once the NLXEXP routine has elicited the data, it should call the special NLSETR routine to pass the translation string and the English string back to the NaturalLink Sessioner. The translation string becomes part of the translation for the sentence, and the English string is placed in the Results window. (The NLSETR routine is provided in the NaturalLink Toolkit software.) NLXEXP then returns to the NaturalLink Sessioner, which proceeds with the query-building process as usual. A step-by-step procedure for implementing user-defined experts follows.

1. When you are using IBUILD to generate the lexicon for an interface, specify a lexical item to be a user-defined expert.

2. Provide an identifying code (an integer) for Lexicon Builder to associate with the lexical item.

3. Complete the interface and generate an interface file.

4. Design the windows that will elicit the user-defined expert data.

5. Write the application routines that will handle the user-defined expert and the NLXEXP routine that the NaturalLink Sessioner will call.

6. Link the application with the NaturalLink object.

If the user, when creating a command sentence from the NLmenu screen, chooses the lexical item that was specified to be a user-defined expert, the following sequence occurs:

1. The NaturalLink Sessioner calls NLXEXP with the identifying code for the expert. (The identifying code for the expert **must** be unique within the application if two or more interfaces are to be loaded or used.)

2. NLXEXP determines by the identifying code which expert is being invoked and calls the application routines that handle that expert.

3. The application routines perform the operations that are necessary to elicit the information, such as reading a file or making calls to Window Manager, and return to NLXEXP.

   Note that the NLmenu screen is displayed throughout the processing of the user-defined expert. All the windows in the application program necessary to elicit the expert information are displayed at the top of the NLmenu screen. If you wish to display the user-defined expert window at all times, place it so that it will not be covered by the NLmenu screen. (User-defined expert windows must be in a separate screen file from those of the NLmenu screen.) Add the user-defined expert window before calling the NaturalLink Sessioner.

4. NLXEXP calls NLSETR to pass the translation string and English text for the Results window back to the NaturalLink Sessioner.

5. NLXEXP returns to the NaturalLink Sessioner.

6. The NaturalLink Sessioner proceeds as usual to invoke the NaturalLink Parser, which determines the choices available next.

The calling sequences for the NLXEXP and NLSETR routines are described in Appendix C, High-Level Language Interface, along with the calling sequence for the NaturalLink Sessioner, NLDRIV.

An example of the algorithms for dealing with user-defined experts follows.

The application main program

```
.
.
Call NLLOAD with the name of the interface file.
Call NLDRIV with the number of the loaded
      interface file.
If error,
  Call the message manager to report the error.
  Exit the program.
.
.
.
```

The NLXEXP routine

```
Call expert routine 1 if the identifying code is 1.
Call expert routine 2 if the identifying code is 2.
Call expert routine 3 if the identifying code is 3.
Call expert routine 4 if the identifying code is 4.
Call NLSETR with the translation string and the
      English text returned from the expert routine
      that was called.
Return to the NaturalLink Sessioner with proper return code.
```

A sample expert routine

```
Load the expert-handling screen.
Read the stock list file and put the stocks
        in the stock list window.
Add and select the stock list window.
Receive the user's choice from the window.
Release the stock list window.
If the choice was some other stock,
  Add and select the specific stock symbol window.
  Receive the user's specific stock symbol.
  Delete the specific stock symbol window.
Delete the stock list window.
Create an English text string containing
        the stock symbol chosen.
Unload the expert-handling screen.
Return to NLXEXP with the translation string and the English
        text string.
```

## Dynamic Lexical Items

**9.3** The Dynamic Lexical Items feature allows for limited modifications to the lexicon and the screen of an interface through an application program. Modifications can be made on the copy of the interface in memory only or saved to disk.

### Uses for Dynamic Lexical Items

**9.3.1** When something about your interface needs to be determined at run time, such as whether or not a printer is available or what fields are in the file being searched, you must modify the options available to the user. In some cases user-defined experts solve the problem completely. In other cases, such as those described later in this chapter, user-defined experts provide a flawed solution at best and sometimes no solution at all. To achieve the ideal solution, you must modify the lexicon directly.

### Possible Modifications

**9.3.2** You can remove terminal elements from the lexicon (and, therefore, from the screen). You can create new items, if you desire, to replace the ones removed. These new items are lexical items that duplicate the original element, but the original element no longer exists. (This situation is similar to that discussed in paragraph 8.7, Create Duplicate Item.) For the rest of this discussion, a terminal element that was removed or is to be removed from the lexicon (the original element) will be called a dynamic terminal symbol (DTS). You can create more than one item to replace each DTS removed, and you can make modifications that remove, add to, or replace items that were created earlier.

### Preparations for Using the Feature

**9.3.3** To use this feature, you must identify the DTS as a user-defined expert when you create the lexicon in IBUILD. The expert code assigned to the DTS will identify to the Sessioner the symbol you wish to remove or replace. Once the interface file has been generated, you must modify the application program by either of two methods:

- Call NLXSND for each DTS that is to be removed or replaced, and call NLXCRE to perform the modifications to the interface in memory and, optionally, to disk.

- Instead of calling NLXSND for each DTS, you can store all the information required to modify the interface in a file and call NLXCRE using the Information File option.

You can choose either method as long as the information is specified. Calling NLXCRE without calling NLXSND first and not using the Information File option produces an error.

**Procedure** **9.3.4** Two calls to the NaturalLink Sessioner handle the creation or deletion of lexical items. The first call, NLXSND, sends the information that tells the Sessioner what items to create or delete.

The second call, NLXCRE, modifies the interface. This call has the following options:

- Load Interface File — NLXCRE can load the interface file to be modified if the application program has not already loaded it via an NLLOAD call.
- Load Information File — You can specify an information file so that the NLXCRE call loads the information that would normally be sent through NLXSND. You can use this option instead of or in addition to the NLXSND call.
- Save Interface — You can save the modified interface file to disk either under the same name as the original interface file or under a different name. You can do this when you modify the interface, or you can do it later by making another NLXCRE call and specifying the Save option.
- Unload Interface From Memory — The modified interface file can be unloaded from memory so that the application does not have to make the NLXUNL call.
- Restore — After making modifications to an interface using NLXCRE, you can restore it to almost its original state by making another NLXCRE call with the Restore option specified. You can do this even after the interface has been saved to disk. The restored interface differs in two ways from its original state:
  - The phrase for the lexical item removed from the lexicon is not returned to the same position in the window it came from but instead becomes the last item.
  - Any duplicate lexical items created through the Lexicographer option in IBUILD and any items created through the dynamic lexical items feature are removed from the lexicon.

When NLXCRE saves an interface to disk, the information for restoring the interface is stored also. The NaturalLink software creates the name for this modifications file by adding a different extension, .ND$, to the name of the file to which the new interface file was written, such as NEWFILE.INT. The result is a name like NEWFILE.ND$. This file must exist with its companion interface if the new interface is to be modified or restored. Therefore, if you rename the new interface file, you must also rename the saved information file as < NEWNAME > .ND$.

IF THE MODIFICATIONS FILE IS LOST, THE MODIFIED INTERFACE CAN NEVER AGAIN BE MODIFIED OR RESTORED. Therefore, the Sessioner sets the read-only attribute for the modifications file so that users cannot delete it accidentally.

The item text for a new lexical entry is added to the end of the list of items for its window. However, if the new item text matches text already in the window, the new item is assigned to the existing text.

If NLXCRE is unable to create a new lexical item because of an invalid window label or a bad user-defined expert code (the DTS code), it returns a warning to the application after creating all the entries that it can. The application can proceed to build commands from this interface, but some of the lexical entries that should have been created will be missing.

## Sending Information to the Sessioner

**9.3.5** The calls described earlier, NLXSND and NLXCRE, pass information to the Sessioner and make the modifications. Both these calls can be used several times within an application program. NLXCRE must not be used while an NLDRIV call is in progress (when the Sessioner gives control to the application to process a user-defined expert or a user-defined function key). The Dynamic Lexical Items feature modifies window item lists; using the feature during an NLDRIV call can cause severe problems in the NaturalLink software.

When you create the interface, you must create a terminal element (the DTS) in the grammar for the new lexical items to replace. You must also perform these tasks when you create the lexicon with the Lexicon Builder option in IBUILD:

1. Assign the DTS to any window.
2. Assign the DTS to a phrase.
3. Make the DTS a user-defined expert and give it a unique expert code, which must be different from the expert code for any other user-defined expert or DTS in this interface.

After the interface file has been generated, the application program does the following:

1. Loads the interface to be modified through the NLLOAD call (or, if you wish, through the NLXCRE call).

2. Gathers the information to create the new lexical items. This information can be stored in a file that the Sessioner loads through the NLXCRE call, or it can be passed directly to the Sessioner using the NLXSND call.

3. Issues the NLXCRE call to create the new lexical items.

In response to the NLXCRE call, the Sessioner does the following:

1. Loads the interface (if it is not already loaded)

2. If an information file was specified, loads the information to create the new lexical items

3. Creates the new lexical items

4. If requested, saves the modified interface to a file and saves the information about the DTS that was replaced in the modifications file associated with this interface

5. If requested, unloads the interface from memory

6. Returns to the application

The syntax of the NLXSND call to send the new lexical items information to the NaturalLink Sessioner is as follows:

`STAT = NLXSND(`*dts_code*`,` *option*`,` *number*`,` *window_label_array*`,`
*item_text_array, trans_array*`)`

All parameters are input only.

| | |
|---|---|
| *dts_code* | Integer; expert code assigned in the Lexicon Builder to the terminal element to be affected. |
| *option* | Integer; operation to be taken with the specified DTS. |

The options are as follows:

| | |
|---|---|
| Restore | (0) Remove all lexical items for this DTS; restore the original entry to the lexicon and the item text to the screen. If the number and array parameters are not zero and null, an error is returned. |
| Delete | (1) Remove the DTS entry from the lexicon and the screen; remove any previously specified lexical entries, and do not replace with anything. If the number and array parameters are not zero and null, an error is returned. |
| Add | (2) Remove the DTS entry from the lexicon and the screen, and add lexical entries from the information in the three arrays. |
| Replace | (3) Remove the DTS entry from the lexicon and the screen; remove any previously specified lexical entries, and add lexical entries from the information in the three arrays. |

| | |
|---|---|
| *number* | Integer; number of entries to be created; should be zero in options 0 (Restore) and 1 (Delete). |
| *window_label_array* | Array of character strings containing the window labels for the lexical entries; should be null in options 0 (Restore) and 1 (Delete). |
| *item_text_array* | Array of character strings containing the item text for the lexical entries; should be null in options 0 (Restore) and 1 (Delete). |
| *trans_array* | Array of character strings containing the translation text for the lexical entries; should be null in options 0 (Restore) and 1 (Delete). |

The three arrays are assumed to line up horizontally (that is, the first entry in each array creates the first lexical entry, the second entry in each array creates the second, and so on).

Appropriate versions of this call exist for languages that do not easily support arrays of character strings. See Appendix C, High-Level Language Interface, for specific information about issuing this call from your language.

## Information File for New Lexical Items

**9.3.6** Following is the format of the information file if the information for new lexical items is to be loaded from the disk.

```
v,<the version of the Toolkit that this file was created for>
n,<# of entries in the file>
c,<DTS code>,<option>,<number of lexical entries to create>
w,<window label>
i,<item text>
t,<translation>
*,<a comment - may be inserted anywhere>
```

This file is a sequential file that can be created with any standard text editor or from a program. The maximum length of a line in this file is 80 characters.

The one-character identifiers followed by a comma at the beginning of each line are called command markers. You can insert comments into the file by preceding them with a command marker of `*,`.

The number of `c,` lines must equal the number of entries specified on the `n,` line.

The `w,`, `i,`, and `t,` lines come in sets. They represent the window label, item text, and translation for each entry that should be created, and they must occur in the order specified. There must be one set of these lines for each entry to be created, asindicated in the *number of lexical entries to create* item on the preceding `c,` line.

A null translation is indicated by a `t,` line with no translation text following the comma.

Example File:

```
v,2.0
n,3
*,  This entry adds 5 fields to whatever fields
*,  already exist for DTS number 12
c,12,2,5
w,FIELDS
i,employee name
t,EMPNAME
w,FIELDS
i,employee address
t,EMPADDR
w,FIELDS
i,job level
t,JOBLVL
w,FIELDS
i,salary
t,EMPSAL
w,FIELDS
i,date hired
t,HIRDAT
*,  This entry restores DTS number 15's original lexical
*,  entry and removes any other lexical entries
c,15,0,0
*,  This entry replaces whatever entries already exist for
*,  DTS number 17 with a new item
c,17,3,1
w,OPTIONS
i,and send the output to a printer
t,@1 /P
```

If any errors are found in the file, NLXCRE returns with an error message indicating the type of error encountered.

## Specifying Key Codes in Translations

**9.3.7** You can enter a key code as part of a translation for lexical items to be created; enter a | character followed by a two-digit hexadecimal key code for the key. If a | character is needed as part of a translation, enter ||. Use this method to specify key codes in translations in the NLXSND call and in the information file.

For example, if a lexical item is to have the translation

AB< carriage return> CD | EF

the translation text must be

AB | 0DCD | | EF

## Duplicate Lexical Items

**9.3.8** Duplicate lexical items created for a DTS through IBUILD and the Lexicographer option are allowed. However, when you specify option 0 (Restore), 1 (Delete), or 3 (Replace) for a DTS with duplicates, all lexical entries for that DTS are stripped from the lexicon. This includes both dynamically created entries and duplicate lexical items created through IBUILD.

## Using Dynamic Lexical Items

**9.3.9** You should modify the lexicon as infrequently as possible. The process can be slow, and in some cases the delay may be unacceptable. The time needed depends on several factors: the size of the interface; the number of lexical entries to be created; and the state of NaturalLink's memory space (the quantity of memory available and the amount of fragmentation). Because various factors are involved, you need to try dynamic lexical items before deciding if you can use them. If you have a large interface, there may not be enough memory to allow creation of lexical items. If there is not enough memory, you can separate the modification process from the query-building process, since there is an option to save the modified interface in a file.

Modified interfaces run as fast as other interfaces during the query-building process, provided memory has not been greatly fragmented by the procedure for creating lexical items. If a modified interface is saved in a file, it can be loaded and used like any other interface. The recommended method of using dynamic lexical items is to modify the interface, save it in a file, flush NaturalLink's memory through the WMFLSH call (or quit and run another process that builds queries), load the modified interface, and run as usual.

WMFLSH clears all screens and interfaces that have been loaded into memory. All load calls must be performed again, and windows created through the WMWCRE call prior to the WMFLSH call must be recreated. Refer to Chapter 6 of the *NaturalLink Window Manager Reference Manual* for more information on the WMFLSH call.

**CAUTION: The Dynamic Lexical Items feature and the Synonyms portion of the Interface Customization feature (see Chapter 16, Interface Customization Feature) both modify the position of items in the interface screen by adding to and subtracting from the window item tables. DO NOT allow the user to save sentences (see Chapter 15, paragraph 15.2.5, Sentences Option) from an interface for which either of these features is provided. Recalling a saved sentence when the interface screen items are not in the same positions WILL CAUSE A SYSTEM CRASH. The Edit Phrases portion of the Interface Customization feature does not modify item positions and can be used in conjunction with the saved sentences feature.**

## Attaching Help to Items Created

**9.3.10** If the DTS item being replaced had Help information attached to it, that Help information is attached to each of the new items created.

## Examples

**9.3.11** The following examples illustrate some of the factors that determine whether you should select user-defined experts or dynamic lexical items to control user options.

Example 1 — Listing Fields in a File

*Problem*: Assume you are writing a NaturalLink interface to a database with such files as an employee information file. Each file has fields within it: employee name, address, job level, salary, and date hired. Here is a sample grammar for querying a file within this database.

```
SENTENCE → open file file_expert
SENTENCE → look_at records of file where QUALIFIER
QUALIFIER → alpha_field {is is_not} alpha_expert
QUALIFIER → num_field COMPARISON_OPERATOR num_expert
QUALIFIER → date_field {of after before} date_expert
COMPARISON_OPERATOR → is_equal_to
COMPARISON_OPERATOR → is_not_equal_to
COMPARISON_OPERATOR → is_greater_than
COMPARISON_OPERATOR → is_less_than
```

This database requires that files be opened before they can be queried, and only one file can be queried at a time. After the user has opened a file, you want to find out from the database the fields in this file and put the fields in a list for the user to choose from. (You do this only if the fields within a file can change at any time or if you do not know about the files and the fields when you create the interface.)

To put the fields in a list by means of user-defined experts, you need three field types: one for alpha_field, one for num_field, and one for date_field. These three field types cannot be combined into one expert, because different comparison operators follow them. The lexical entries for these three experts would look like this:

| | | |
|---|---|---|
| terminal element | — | alpha_field |
| | | user-defined expert 1 |
| window label | — | FIELDS |
| item text | — | <*specific alpha field*> |
| terminal element | — | num_field |
| | | user-defined expert 2 |
| window label | — | FIELDS |
| item text | — | <*specific numeric field*> |
| terminal element | — | date_field |
| | | user-defined expert 3 |
| window label | — | FIELDS |
| item text | — | <*specific date field*> |

Since all three experts will be available at the same time, the user must choose the type of field to query on before selecting the field.

If there are no date fields in the opened file, the following happens:

1. The user chooses < *specific date field* > .

2. A message appears saying that no date fields are in this file.

3. The user must go back and choose a different field type.

*Solution*: Use dynamic lexical items. When you find out from the database what fields and field types the file contains, you can create lexical items to replace the lexical entries for alpha_field, num_field, and date_field. Your lexicon looks the same as when you specify the information in IBUILD. After you create the new lexical items, the lexicon looks like this:

| | |
|---|---|
| terminal element | — alpha_field |
| translation string | — EMPNAME |
| window label | — FIELDS |
| item text | — employee name |

| | |
|---|---|
| terminal element | — alpha_field |
| translation string | — EMPADDR |
| window label | — FIELDS |
| item text | — employee address |

| | |
|---|---|
| terminal element | — num_field |
| translation string | — JOBLVL |
| window label | — FIELDS |
| item text | — job level |

| | |
|---|---|
| terminal element | — num_field |
| translation string | — EMPSAL |
| window label | — FIELDS |
| item text | — salary |

| | |
|---|---|
| terminal element | — date_field |
| translation string | — HIRDAT |
| window label | — FIELDS |
| item text | — date hired |

This example assumes that the database under discussion supports both a descriptive field name and the actual name assigned to the field (that is, *employee name* as well as EMPNAME). The descriptive name is not required, but it looks nicer to the user.

Now when the user wants to choose a field, he or she is presented with a list of the fields in the file, rather than with a list of types of fields.

*An Additional Benefit*: When a terminal element in the grammar is an expert, it can have no slots ("@1") in its translation string. In the target language translation, another terminal element's translation string must contain a slot into which the expert can be plugged. There can be as many slots as desired in the translation. The application controls the translation by using the Dynamic Lexical Items feature to replace an expert in the lexicon with items that have translations. For example, EMPNAME might be given the translation "EMPNAME@1@2."

You need not assign the newly created lexical items to the same window as the DTS they replace. You can assign them to any parse window in the interface screen.

*Call Sequence*: Assume the following lexical entry is to be modified, as well as the three field entries:

| | |
|---|---|
| terminal element — | look_at |
| | user-defined expert 4 |
| window label — | OPERATIONS |
| item text — | look at |

This lexical modification will allow the application to force the user to open a file before looking at the information.

Following is the sequence of calls for making modifications to the interface for Example 1:

```
Call NLLOAD to load the interface.
Until the user quits,
   If no file is open,
      Call NLXSND and NLXCRE to remove the look_at terminal element
         from the lexicon so the user cannot choose it.
   Call NLDRIV to allow the user to build a command.
   Execute the command.
   If the command was to open a file,
      Find out what fields are available for the opened file.
      For each type of field (alpha, numeric, or date),
         Call NLXSND, specifying the information for creating a
            lexical entry for each field of that type.
         Call NLXSND to fill in the appropriate lexical entry for the
            look_at terminal element.
         Call NLXCRE to create the new lexical entries.
         Set a flag indicating that a file is open.
   If the command was to close a file,
      Set the flag indicating that there are no files open.
When the user quits, call NLXUNL to unload the interface file.
Quit.
```

Example 2 — Creating Optional Interfaces

*Problem*: Assume you are creating a NaturalLink interface to a software package that can send data to a printer if one is attached to the user's terminal. Here is a sample grammar rule:

S $\longrightarrow$ display data (and_send_to_printer)

When you create the lexicon through IBUILD, make the following entry:

| | |
|---|---|
| terminal element — | and_send_to_printer |
| | user-defined expert 5 |
| window label — | OPTIONS |
| item text — | and send it to the printer |

When your application is installed, it asks the user if a printer is attached to the user's machine. If one is attached, the option to send data to the printer should be available; if one is not attached, the option should not be available. However, your application has no way of changing the grammar, and the grammar says that the option is always available.

*Solution*: When your application is installed and you find that a printer is not available, use the Dynamic Lexical Items feature to delete the item from the lexicon. You need not create a replacement. If a printer is available, use dynamic lexical items to create a lexical entry with the appropriate translation. For example:

| | |
|---|---|
| terminal element | — and_send_to_printer |
| translation string | — @1 /p |
| window label | — OPTIONS |
| item text | — and send it to the printer |

This technique also works well for optional disk drives.

## Example 3 — Limiting Path Availability

*Problem*: Assume you have a path in your grammar that is to be available only under specific circumstances. For example, only certain people are allowed to create files on your database. Following is the sample grammar:

S $\longrightarrow$ print REPORT
S $\longrightarrow$ find QUERY_SPECIFICATION
S $\longrightarrow$ create FILE

Without dynamic lexical items, you have two choices:

- Create a different interface file for those who are not allowed to create files so that the Create path is not available.

- Allow all users to build a Create command and prevent unauthorized users from executing it.

*Solution*: Using the Dynamic Lexical Items feature, delete the Create item from the lexicon when users not authorized to create files start the application. Since the option will no longer exist, they will be unable to select it.

**Sample Calls** **9.3.12** The following paragraphs provide sample calls you use with the Dynamic Lexical Items feature. These calls send new lexical items information to the Sessioner.

***Remove All Existing Entries and Create New Entries*** **9.3.12.1** This sample call indicates that all existing entries for DTS 23 should be removed and five new ones created. This call specifies the Replace option.

```
STAT = NLXSND(23, 3, 5, WINDOW_ARRAY, ITEM_ARRAY, TRANS_ARRAY);
if (STAT != 0)
{
  printf("error %d from NLXSND\n",STAT);
  exit();
}
```

***Create New Entries and Leave Existing Entries*** **9.3.12.2** This sample call creates five new entries for DTS 23 and leaves existing entries (except for the DTS, which is removed if it still exists). It specifies the Add option.

```
STAT = NLXSND(23, 2, 5, WINDOW_ARRAY, ITEM_ARRAY, TRANS_ARRAY);
if (STAT != 0)
{
  printf("error %d from NLXSND\n",STAT);
  exit();
}
```

***Restore Original DTS Entry and Remove All Others*** **9.3.12.3** This sample call restores the original DTS entry and removes all other entries. It specifies the Restore option. Note that the number of entries to create is zero and the three array parameters are null.

```
STAT = NLXSND(23, 0, 0, NULL, NULL, NULL);
if (STAT != 0)
{
  printf("error %d from NLXSND\n",STAT);
  exit();
}
```

***Remove All Entries*** **9.3.12.4** This sample call indicates that all entries for DTS 23 (including the DTS) should be removed and not replaced. It specifies the Delete option. Again, the number of entries to create is zero and the three array parameters are null.

```
STAT = NLXSND(23, 1, 0, NULL, NULL, NULL);
if (STAT != 0)
{
  printf("error %d from NLXSND\n",STAT);
  exit();
}
```

**CAUTION: Be extremely careful not to remove a symbol and replace it with nothing (delete it) unless other items will be available at the same time. Doing this can cause the system to crash. As an example, assume that the following two rules are the only rules in your grammar.**

**X $\longrightarrow$ A B (C)**
**X $\longrightarrow$ D E**

**You can delete C because it is optional. You can delete either A or D because the other could be chosen instead, but do not delete both: no selections would be available. Neither B nor E can be deleted unless the preceding element is deleted also; if A is chosen, B is the only possible choice, and if D is chosen, E is the only possible choice. You can also replace any symbol with something else. DELETING A SYMBOL THAT IS THE ONLY POSSIBLE CHOICE AT A GIVEN POINT IN THE GRAMMAR WILL CREATE A SITUATION IN WHICH NO WINDOWS ARE ACTIVE FOR SELECTION BUT THE QUERY IS NOT EXECUTABLE.**

***Create New Lexical Items***

**9.3.12.5** This is the syntax for the NLXCRE call to create new lexical items:

`STAT = NLXCRE(`*int_file*`,` *int_number*`,` *info_file*`,` *out_file*`,` *unload*`,` *option*`)`

| | |
|---|---|
| *int_file* | Input; a character string; the name of the interface file to be loaded. This should be null if the interface file is already in memory; if it is not null and the interface is already in memory, the interface will not be reloaded. |
| *int_number* | Input/output; the address of an integer; the number of the interface as returned from a NLLOAD call. If *int_file* is not null and the interface is not already in memory, the initial value is ignored and the value is set to the interface number to be used in subsequent NLDRIV or NLXUNL calls (unless *unload* is specified; if *unload* is specified, the value is set to −1). |
| *info_file* | Input; a character string; the name of the file that contains information for creating the lexical items. If the information was previously sent through the NLXSND call, both sets of information are used in creating new lexical items. |

| | |
|---|---|
| *out_file* | Input; a character string; the name of the interface file to be created. The name can be the same as that of the original interface file or different. The value should be null if the interface is not to be saved at this time. |
| *unload* | Input; an integer; this is ignored if *out_file* is null. The options are:<br>0—Leave the interface in memory.<br>1—Unload the interface before returning to the application. |
| *option* | Input; an integer. The values are: |

| | |
|---|---|
| None | (0) No special options. |
| Save Only | (1) This call is saving an interface already modified by a previous NLXCRE call. All parameters except *int_number*, *out_file*, and *unload* are ignored. |
| Restore | (2) This call is restoring the interface to its original state. *Info_file* parameter is ignored, and any DTS information about this interface is deleted from memory. |
| Clear | (3) This call is clearing any information sent earlier with the NLXSND call. All other parameters are ignored; the interface in memory is not affected. |

NLXCRE uses all modification information sent by NLXSND after the last NLXCRE call that modified the interface. An NLXCRE call with the option parameter set to 1 (Save Only) does not modify the interface; therefore, the modification information is not cleared. If you wish to clear from memory modification information that has been sent to the Sessioner, make an NLXCRE call with the option parameter set to 3 (Clear).

#### *Information Sent; Leave Interface in Memory*

**9.3.12.6** This sample call indicates that the interface is loaded, the information has been sent through NLXSND, and the interface is not to be saved into a file and should be left in memory. The call actually applies the information sent by NLXSND. Since the *out_file* parameter is null, modifications are made only in memory.

```
STAT = NLXCRE(NULL, &INT_NUMBER, NULL, NULL, 0, 0);
if (STAT != 0)
{
  printf("error %d returned from NLXCRE\n",STAT);
  exit();
}
```

***Load Dynamic Lexical Items, Save Interface, and Unload Interface***

**9.3.12.7** This sample call indicates that the interface has not been loaded, the dynamic lexical items information should be loaded from an information file, and the interface is to be saved into a file and unloaded from memory.

```
STAT = NLXCRE("file.int", &INT_NUMBER, "info.dat", "newfile.int",
     1, 0);
if (STAT != 0)
{
  printf("error %d returned from NLXCRE\n",STAT);
  exit();
}
```

***Unload Interface From Memory and Save to Disk***

**9.3.12.8** This sample call indicates that the interface is already loaded and modified, and that it should be saved to disk and unloaded from memory; no new modifications are required.

```
STAT = NLXCRE(NULL, &INT_NUMBER, NULL, "newfile.int", 1, 1);
if (STAT != 0)
{
  printf("error %d returned from NLXCRE\n",STAT);
  exit();
}
```

***Restore DTS Entries, Remove Lexical Items, and Save Interface to Disk***

**9.3.12.9** This sample call indicates that the interface is already loaded and that all DTS entries should be restored, all lexical items created by the application should be removed, and the resulting interface should be saved to disk but not unloaded from memory.

```
STAT = NLXCRE(NULL, &INT_NUMBER, NULL, "newfile.int", 0, 2);
if (STAT != 0)
{
  printf("error %d returned from NLXCRE\n",STAT);
  exit();
}
```

***Clear From Memory Information Sent to Sessioner***

**9.3.12.10** This sample call indicates that all information sent to the Sessioner through the NLXSND call is to be cleared from memory.

```
STAT = NLXCRE(NULL, &INT_NUMBER, NULL, NULL, 0, 3);
if (STAT != 0)
{
  printf("error %d returned from NLXCRE\n",STAT);
  exit();
}
```

# GENERATING NATURALLINK SENTENCES 10

## Introduction

**10.1** After your lexicon is complete, a new option—Generate Representative Sentences for the Interface—appears on the Create and Test Interface main screen. Use this option to test the sentence types your grammar produces.

---

**NOTE:** Two other options—List the Lexicon for the Interface (discussed in Chapter 14) and Check the Screen Against the Lexicon for the Interface (discussed in Chapter 11)—are also now available. If they are not available, press the **F6** key to check the status of your lexicon.

---

When you select the Generate Representative Sentences option, the NaturalLink sentence generator utility tests all paths through your grammar and produces a set of sentences representative of those paths. This option is important because even though your grammar passed the initial grammar tests—Format, Static Well-Formedness, Common Expansions, Inconsistent Translations, and Infinite Parses—it may not produce all the sentences, and only the sentences, required by your application program. When you examine the generated sentences, you can see certain errors that the grammar tests could not find. These errors include superfluous sentences, semantically incorrect sentences, sentences in which the word order is wrong, and sentences that are missing certain contextually important words or phrases.

## Procedure

**10.2** The Generate Representative Sentences option loads and processes the necessary rule information, generates a representative set of sentences, and places those sentences in an output file. The following filename is assigned to the output file:

< *interface file* > .NG$

where:

< *interface file* > is the first part of the name (without its extension) that you designated initially as the interface filename. (For example, if you designated JOBSHOP.INT as the interface filename, the system would name the generated sentence file JOBSHOP.NG$.)

The generated sentences have the following format:

```
!<string of integers for sentence 1>
#sentence 1

!<string of integers for sentence 2>
#sentence 2
...
```

where:

! is the flag indicating a string of integers.

# is the flag indicating a sentence.

The string of integers represents a mapping of the lexical items and is used by the Translation Test. The sentence string represents the actual sentence. You can look at this sentence file using the View a File function key discussed in Chapter 14, Other Interface Builder Options. Here is an example of the sentences in a typical generated sentences file.

*Representative Sentences*

```
!1 7 13 19 15
# Find workers whose name is equal to
  < specific name >

!1 7 16 19 17
# Find workers whose employee number is equal to
  < specific number >

!1 7 13 19 15 18 16 19 17
# Find workers whose name is equal to < specific name >
  and whose employee number is equal to < specific number >

!3 10 5 7 13 19 15
# Find name of workers whose name is equal to
  < specific name >

!3 10 5 7 16 19 17
# Find name of workers whose employee number is
  equal to < specific number >
```

If an examination of the generated sentences file reveals problems in your grammar, you must correct the grammar and retest it, modifying your lexicon if necessary. Then run the NaturalLink sentence generator utility again to produce a new set of representative sentences. Repeat this procedure until the sentences generated represent all the sentences, and only the sentences, required by the application program.

# CHECKING AN INTERFACE SCREEN AGAINST ITS LEXICON 11

## Introduction

**11.1** After the NaturalLink lexicon is complete, an additional option, Check the Screen Against the Lexicon for the Interface, becomes available in the Create and Test Interface main screen. This screen/lexicon consistency check tests whether all windows referenced by the lexicon are, in fact, in the screen file specified for the interface, and whether the windows have been designed correctly with regard to their attributes and functions. You can perform this test either before or after selecting the Generate Representative Sentences for the Interface option (described in Chapter 10, Generating NaturalLink sentences). The screen/lexicon consistency check is *not* optional; it must be run or you cannot generate the interface file.

When you select the Check the Screen Against the Lexicon option, a pop-up window appears requesting the name of a message output file. You can view this file after the test by pressing the View a File function key described in Chapter 14, Other Interface Builder Options.

The Check the Screen Against the Lexicon option returns two types of messages:

- Error messages — Point out error conditions in the screen file that must be corrected before you can generate the interface file.
- Warning messages — Point out error conditions in the screen file that should be corrected as soon as possible. The test will not fail as a result of these error conditions. The Generate Interface File routine corrects these errors in the interface file but not in the screen file. You should correct the screen file at the earliest opportunity.

## Types of Messages

**11.2** The following paragraphs discuss the two types of messages returned by the Check the Screen Against the Lexicon option.

### Error Messages

**11.2.1** The following errors cause the consistency check test to fail and cause the appropriate error message to be sent to the specified output file.

- A window referenced by the lexicon is missing from the screen file.
- A Command window is not specified for the lexicon.
- A Results window is not specified for the lexicon.
- An Execute item is not specified in the Command window.
- The window identified as the Command window is no longer in the screen file.

- The window identified as the Results window is no longer in the screen file.
- Lexical items are specified for the Command or Results windows.
- An inaccurate item label is specified for a command or function key in the Command window. Refer to Chapter 15, The NaturalLink Sessioner, for information on item labels.

## Warning Messages

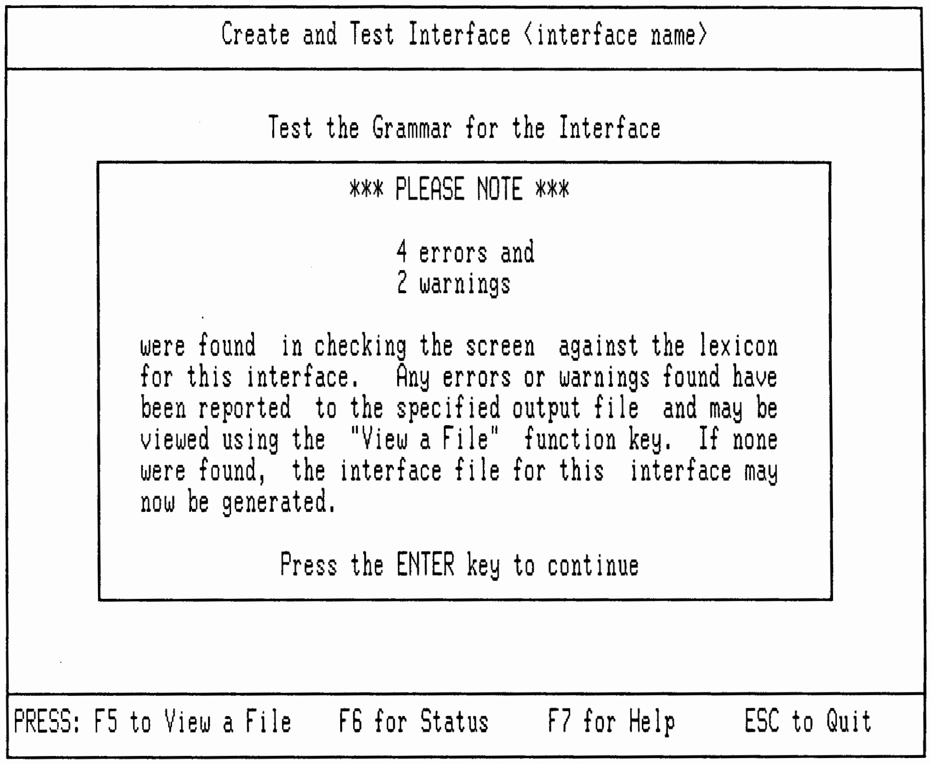
**11.2.2** The following errors are reported as warning messages. The entries also identify the measure that will be taken to correct the error in the interface file.

- A window containing lexical items is not defined as a list window or a user-defined window. It will be changed to a list window.
- The Command window is not defined as a list window or a user-defined window. It will be changed to a list window.
- The Results window is defined either as an edit window or a file window. It will be changed to a display window. Text, list, and user-defined windows are also valid types for the Results window.
- A lexical item is not already in the window to which it was assigned. The item will be added to the end of that window's item table.
- Any of the following attributes is set to 1 (Yes) for a window containing lexical items, or these attributes are set to 1 (Yes) for the Command window: Multiple Selection, Special Repaint on Receive, or Don't Redisplay Current Items. The appropriate attributes will be reset to 0 (No).
- Either of the following attributes is set to 1 (Yes) for the Results window: Multiple Selection or Special Repaint on Receive. The appropriate attributes will be reset to 0 (No).

## Viewing an Error File

**11.2.3** Figure 11-1 shows an example of the pop-up window that appears after the screen/lexicon consistency check has been run. To look at the output file, press the View a File function key from the Create and Test Interface main screen and enter the name of the output file.

**Figure 11-1** **Sample Consistency Check Error Screen**



```
                Create and Test Interface <interface name>

                     Test the Grammar for the Interface

                            *** PLEASE NOTE ***

                               4 errors and
                               2 warnings

          were found  in checking the screen  against the lexicon
          for this interface.  Any errors or warnings found have
          been reported  to the specified output file  and may be
          viewed using the  "View a File"  function key.  If none
          were found,  the interface file for this  interface may
          now be generated.

                      Press the ENTER key to continue

PRESS: F5 to View a File    F6 for Status    F7 for Help    ESC to Quit
```

# SPECIFYING HELP AND WINDOW COORDINATES FOR SESSIONER WINDOWS 12

## Introduction

**12.1** The Specify Help and Window Coordinates for Sessioner Windows option appears in the Create and Test Interface main screen after you have checked your screen against the lexicon and fixed any problems. This option enables you to attach Help messages to certain windows controlled by the Sessioner that you think the user may need help with. This option also enables you to position these windows anywhere on the screen, so that you can avoid covering up important information. The windows to which this option applies are for the Interface Customization feature (see Chapter 16, Interface Customization Feature) and the Saved Sentences feature (see Chapter 15, The NaturalLink Sessioner).

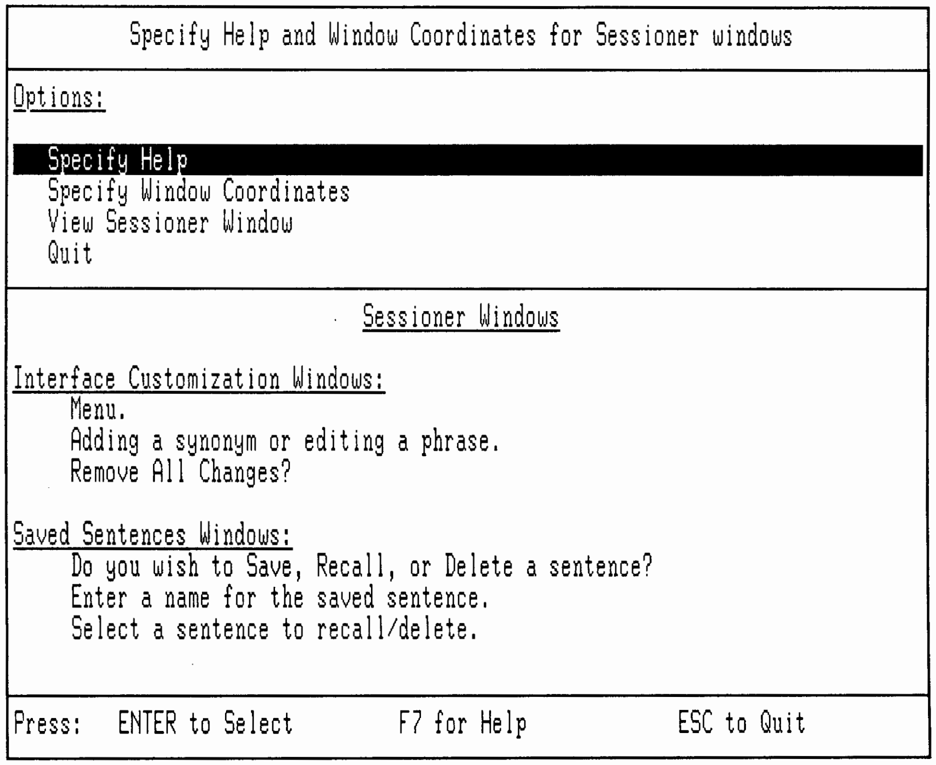
## Procedure

**12.2** If you want to use the Specify Help and Window Coordinates for Sessioner Windows option, you must select it before generating the interface file, because any information specified will be stored in that file. The information specified is permanently stored in the expert information file (< INTFILE.NP$ >); therefore, this file and the history file are updated when you use this option.

### Choosing Windows

**12.2.1** When you choose the Specify Help and Window Coordinates for Sessioner Windows option, the screen shown in Figure 12-1 appears. You are prompted for the information you want to specify and given a choice of windows. These are the windows created and used by the NaturalLink Sessioner; each has a descriptive label to indicate which window is referenced. If you are not sure which window is referenced, you can use the Help feature for more information.

**Figure 12-1** **Specify Help and Window Coordinates Screen**



```
           Specify Help and Window Coordinates for Sessioner windows

Options:

   Specify Help
   Specify Window Coordinates
   View Sessioner Window
   Quit

                              Sessioner Windows

Interface Customization Windows:
     Menu.
     Adding a synonym or editing a phrase.
     Remove All Changes?

Saved Sentences Windows:
     Do you wish to Save, Recall, or Delete a sentence?
     Enter a name for the saved sentence.
     Select a sentence to recall/delete.

Press:   ENTER to Select         F7 for Help            ESC to Quit
```

Each of the options (except Quit) allows you to select one of the Sessioner windows from the list.

## Description of Sessioner Windows

**12.2.2** Following is a description of the Sessioner windows you can select.

### *Interface Customization Windows*

**12.2.2.1** The Interface Customization Windows are as follows:

- Menu — This is the "I want to" window that contains a list of options available to the user. It is displayed when the user selects the Interface Customization feature described in Chapter 16.
- Adding a Synonym or Editing a Phrase — This window prompts the user to enter a synonym or phrase. It is displayed when the user selects either Add Synonyms or Edit Phrases from the Menu window.
- Remove All Changes? — This window prompts the user to remove or leave all synonyms and edited phrases; it can be answered Yes or No. It is displayed when the user selects Remove All Synonyms and Edited Phrases from the Menu window.

### *Saved Sentences Windows*

**12.2.2.2** The Saved Sentences windows are as follows:

- Do You Wish to Save, Recall, or Delete a Sentence? — This window prompts the user to save, recall, or delete a sentence. It is displayed when the user selects the Sentences option.
- Enter a Name for the Saved Sentence — This window prompts the user to enter a name for the current command sentence to be saved. It is displayed when the user selects the Sentences option and chooses to save a sentence.
- Select a Sentence to Recall/Delete — This window prompts the user to choose a sentence to either recall or delete. It is displayed when the user selects the Sentences option and chooses to recall or delete a sentence that was previously saved.

## Specify Help

**12.2.3** If you choose the Specify Help option from the Specify Help and Window Coordinates screen, you are prompted for specific Help numbers. Enter the identifying numbers (the limit is 10) of the Help message(s) you wish to associate with the chosen window. These Help messages are created with Screen Builder or Message Builder and must be in the Help message file attached to the interface screen.

The order in which you specify Help messages is the order in which they will be presented to the user. Press the **F10** key to commit the numbers you have entered, or press the **ESC** key to abort this operation without committing any numbers you have entered. Use the **DEL** key to delete numbers you no longer want. After pressing **F10** or **ESC**, youare returned to the Sessioner Windows menu to select the next window for which you wish to specify Help. Press **ESC** to return to the Options window.

## Specify Window Coordinates

**12.2.4** If you choose Specify Window Coordinates, you are prompted for specific window coordinates. Enter the top row and left column for the area where you want the window to appear on the NLmenu screen. Press the **ESC** key if you wish to abort this operation without modifying the coordinates that have already been specified. If you do not specify coordinates, the Sessioner centers the window on the screen, using default coordinates.

**View Sessioner Window**

**12.2.5** If you choose to view a window, the interface screen appears first, then the Sessioner window you selected. The Sessioner window's position depends on the coordinates designated for the window; the window is displayed in this same position during interface execution.

Once the window has been displayed, press the **ENTER**, **F10**, or **ESC** key to return to the Sessioner Windows menu. You can then select another window to view or press the **ESC** key to return to the Options window.

You cannot view Help while viewing windows. You can view Help after the interface is generated, using the demonstration program (NLLCDEMO.C) provided with the Toolkit package.

**Quit**

**12.2.6** To quit the Specify Help and Window Coordinates for Sessioner Windows option, choose the Quit option or press the **ESC** key from the Options window. Press **ESC** twice to quit from Sessioner Windows.

# GENERATING THE INTERFACE FILE AND TESTING TRANSLATIONS 13

## Introduction

**13.1** After you have successfully performed the screen/lexicon consistency check described in Chapter 11, a new option—Generate the Interface File—appears on the Create and Test Interface main screen. This utility compiles the grammar and lexicon files and stores them, along with the screen description, in a binary interface file. When this task is complete, another option—Test the Interface's Translations—appears on the Create and Test Interface main screen. This utility uses the newly generated interface file and a sentence file to test the translations specified in your grammar and lexicon files. The sentence file can be one of the following files:

- The representative sentence file described in Chapter 10, Generating NaturalLink Sentences

- A file containing a subset of those sentences

- A sentence file you have developed

## Generating the Interface File

**13.2** The Generate the Interface File option requires no interaction. When you select this option, the system processes the necessary rule information from the grammar and lexicon files and produces the binary interface file. (You specified this file in the Record a New Interface's Information pop-up window described in Chapter 6, Using the NaturalLink Interface Builder.) You must regenerate this file each time you change any of the following:

- The NLmenu screen

- The lexicon

- The grammar

When the interface file is generated, the windows in the NLmenu screen are reordered. For the specific ordering, see Chapter 5, paragraph 5.4, Window Order.

---

**CAUTION: After executing the Generate the Interface File utility, you should exit from the Create and Test Interface screen to the main IBUILD screen before you execute this utility a second time. Failure to do so could cause a system crash due to memory fragmentation.**

---

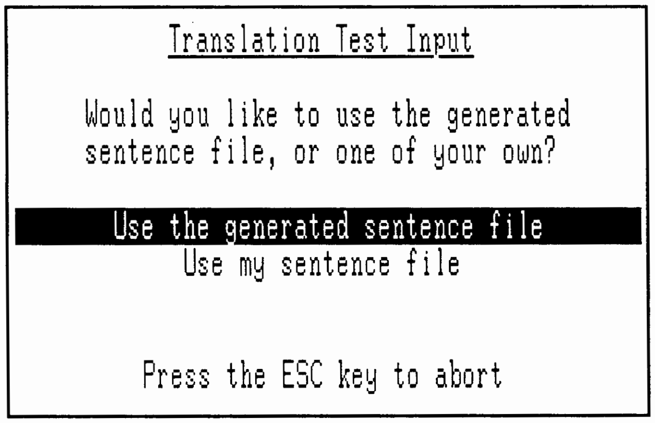
## Testing Translations

**13.3** The Translations Test enables you to test the translations specified by your grammar. While this test is optional, it can save time by helping you locate errors before you produce the user interface. You can perform this test with sentences representative of the whole grammar or focus on a subset. Working with representative sentences is helpful in initial testing to locate problem areas or in verifying that changes made to translation information do not adversely affect the interface. Focusing on a particular subset of sentences is useful in working out interface bugs in isolated areas.

### Using the Generated Sentence File

**13.3.1** When you select Test the Interface's Translations, the pop-up window shown in Figure 13-1 prompts for the type of input you wish to use for the Translation Test. This pop-up window appears only if you have previously run the NaturalLink sentence generator utility described in Chapter 10, Generating NaturalLink Sentences. The translation-testing utility uses the generated sentence output file to produce the NaturalLink translations. If there is no system-generated sentence file, you will be prompted for a sentence file that you must generate.

**Figure 13-1** **Translation Test Input Menu**


Translation Test Input

Would you like to use the generated sentence file, or one of your own?

Use the generated sentence file
Use my sentence file

Press the ESC key to abort

The cursor is on the Use the Generated Sentence File option. If you select this option, you are prompted for an output filename to which the Translation Test results will be sent.

The translation-testing utility uses as the input file the sentence file designated by the system during the Generate Representative Sentences phase —< INTFILE.NG$ >. The system loads the necessary rule information and processes each sentence according to the translation information specified in your grammar and lexicon files. The results of the Translation Test are placed in the designated output file. If this file already exists, you can choose to append the Translation Test results to the existing file or replace the existing file contents with the new results.

The output file, which you can see using the View a File function key, lists the sentences and the results of the Translation Test of each sentence. The test results have the following format:

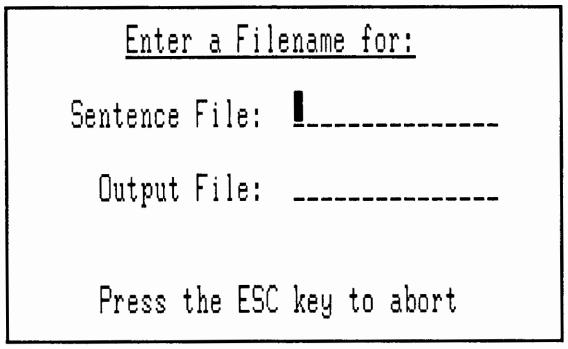
```
sentence:  <sentence1>
  no input translation, translation generated:
  < target language translation>
```

The `no input translation` message appears because no translations were provided in the generated sentence file that was used as input. The `translation generated` message indicates that the system generated the target language translation.

**Testing a Subset**

**13.3.2** If you want to run the Translation Test on a subset of the system-generated sentences, exit from the Interface Builder utility and make a copy of the system-generated sentence file. Using a standard text-editing program, delete the sentence string and the corresponding integer string for each sentence you do not want tested in the copied file. The system can generate translations from either of these strings, so you must delete both. Rerun the Translation Test with this modified file by selecting the Use My Sentence File option in the Translation Test Input pop-up window. Another pop-up window (Figure 13-2) prompts you for the name of your sentence file and for an output file to which the Translation Test results will be sent.

**Figure 13-2** **Testing a Subset of Generated Sentences**



Enter a Filename for:

Sentence File: ______________

Output File: ______________

Press the ESC key to abort

If the output file already exists, you can choose to append the results to the existing file or replace the contents of the existing file with the new results.

If you know what types of sentences your corrected grammar will produce, you can also copy the generated sentence file, edit the copy, and run the Translation Test on the edited sentences. If you decide on this course, be sure to delete the integer string.

There are two reasons for this: (1) the integer string does not match the new lexicon, and (2) if both strings are in the file, the system ignores the sentence string and uses the integer string as input for developing the target language translations. Therefore, any changes you have made to the sentence file *will not* be reflected in the Translation Test output. If you have deleted the integer string, however, the system uses the sentence string as input for developing the target language translations. Thus, your changes to the sentence string *will* be reflected in the Translation Test output.

---

**CAUTION: Do *not* attempt to modify the integer string. Any modification to the integer string could cause the system to crash.**

---

## Using Your Own Sentence File

**13.3.3** Another alternative in translation testing is to run the test on a sentence file you have produced. This procedure is useful for finding specific problems or for isolating errors that would not appear in a file of representative sentences. You can create your sentence file using any standard text-editing program. It must have the following format:

```
#input sentence 1
| translation for input sentence 1 (optional)

#input sentence 2
| translation for input sentence 2 (optional)
...
```

where:

# is the flag indicating a sentence.
| is the flag indicating a translation.

The translation associated with a sentence is optional. If you add an expected translation, the system compares its results with the results you have indicated and reports its findings. If you do not add an expected translation, the system reports the target string it generated from the input.

Next, select the Use My Sentence File option from the Translation Test Input menu (Figure 13-1). A pop-up window prompts you for the name of your sentence file and for an output file in which the Translation Test results will be placed. If the output file already exists, you can choose to append the results to the existing file or replace the contents of the existing file with the new results.

**Errors Reported**

**13.3.4** The following types of errors are reported by the Translation Test.

- System encountered a word or phrase for which no lexical item exists (includes misspelled words).
- System could not execute the input sentence (the grammar does not generate the sentence).
- Input translation does not match generated translation.

---

**NOTE:** Each time you modify the grammar, you must run each step of the Create and Test an Interface procedure again, starting with the five grammar tests.

---

## Translation Test Results

**13.4** Two examples of Translation Test results follow.

### Example 1

**13.4.1** Following is an example of the Translation Test results obtained from a file of representative system-generated sentences. Four listings are presented to show how the Translation Test results were derived:

- Grammar
- Lexicon
- Representative Sentences
- Translation Test Results

Grammar

```
@S find WORKER_NP
      ;(1 2)(1 2)!

@S find_attr WORKER_ATTR_CONSTR of WORKER_NP
      ;(1 2 3 4)(1 (3 2 4))!

@WORKER_NP workers
      ;(1)(1)!

@WORKER_NP mod_workers WORKER_MOD_CONSTR
      ;(1 2)(1 2)!

@WORKER_ATTR worker_name
      ;(1)(1)!

@WORKER_ATTR worker_employee_number
      ;(1)(1)!

@WORKER_MOD whose_worker_name_is COMPARISON_PRED
            worker_name_expert
      ;(1 2 3)(1 2 3)!

@WORKER_MOD whose_worker_employee_number _is
            COMPARISON_PRED worker_employee_number_expert
      ;(1 2 3)(1 2 3)!

@WORKER_ATTR_CONSTR WORKER_ATTR (and
WORKER_ATTR_CONSTR)
      ;(1)(1) ;(1 2 3)(2 1 3)!

@WORKER_MOD_CONSTR WORKER_MOD (and
WORKER_MOD_CONSTR)
      ;(1)(1) ;(1 2 3)(2 1 3)!

@COMPARISON_PRED equal_to
      ;(1)(1)!
```

Lexicon

| Terminal Element Lexical Item | Window | English Phrase | Translation |
|---|---|---|---|
| and | CONNECTORS | and | @1 and @2 |
| equal_to | COMPARISONS | equal to | = |
| find | ACTIONS | Find | select * from @1 |
| find_attr | ACTIONS | Find | select @1 |
| mod_workers | NOUNS | workers | workers where @1 |
| of | CONNECTORS | of | @1 from @2 |
| whose_worker _employee _number_is | QUALIFIERS | whose employee number is | employee number @1 @2 |
| whose_worker _name_is | QUALIFIERS | whose name is | name @1 @2 |
| worker_employee _number | FEATURES | employee number | employee number |
| worker_employee _number_expert | ATTRIBUTES | < *specific no.*> | num type-in < expert text > |
| worker_name | FEATURES | name | name |
| worker _name_expert | ATTRIBUTES | < *specific name* > | alpha type-in < expert text > |
| workers | NOUNS | workers | workers |

This grammar and lexicon produce the representative sentences shown next during the Generate Representative Sentences for the Interface phase.

## Representative Sentences

```
(1)  ! 1 6
     #    Find workers

(2)  ! 1 7 13 19 15
     #    Find workers whose name is equal to < specific name >

(3)  ! 1 7 16 19 17
     #    Find workers whose employee number is equal to < specific
          no. >

(4)  ! 1 7 13 19 15 18 16 19 17
     #    Find workers whose name is equal to < specific name > and
          whose employee number is equal to < specific no. >

(5)  ! 3 10 5 6
     #    Find name of workers

(6)  ! 3 11 5 6
     #    Find employee number of workers

(7)  ! 3 10 18 10 5 6
     #    Find name and name of workers

(8)  ! 3 10 5 7 13 19 15
     #    Find name of workers whose name is equal to < specific
          name >

(9)  ! 3 10 5 7 16 19 17
     #    Find name of workers whose employee number is equal to
          < specific no. >

(10) ! 3 10 5 7 13 19 15 18 16 19 17
     #    Find name of workers whose name is equal to < specific
          name > and whose employee number is equal to < specific
          no. >
```

The seventh sentence above reads "Find name and name of workers" because it was generated from the recursive grammar rule

```
@WORKER_ATTR_CONSTR WORKER_ATTR
(and WORKER_ATTR_CONSTR)
```

and the grammar rule

```
@WORKER_ATTR worker_name.
```

The Translation Test produces the output shown next from the system-generated sentence file.

Translation Test Results

```
                    ***** begin Translation Test *****

sentence: Find workers
          no input translation, translation generated:
          select * from workers

sentence: Find workers whose name is equal to < specific name >
          no input translation, translation generated:
          select * from workers where name = < expert text >

sentence: Find workers whose employee number is equal to
                    < specific no. >
          no input translation, translation generated:
          select * from workers where employee
                    number = < expert text >

sentence: Find workers whose name is equal to
                    < specific name >  and whose employee number
                    is equal to < specific no. >
          no input translation, translation generated:
          select * from workers where name = < expert text >
                    and employee number = < expert text >

sentence: Find name of workers
          no input translation, translation generated:
          select name from workers

sentence: Find employee number of workers
          no input translation, translation generated:
          select employee number from workers
```

```
sentence: Find name and name of workers
          no input translation, translation generated:
          select name and name from workers

sentence: Find name of workers whose name is equal to
                    < specific name >
          no input translation, translation generated:
          select name from workers where name = < expert text >

sentence: Find name of workers whose employee number
                    is equal to < specific no. >
          no input translation, translation generated:
          select name from workers where employee
                    number = < expert text >

sentence: Find name of workers whose name is equal to
                    < specific name > and whose employee number is
                    equal to < specific no. >
          no input translation, translation generated:
          select name from workers where name = < expert text >
                    and employee number = < expert text >

                    ----- Translation Test complete -----
```

After the Translation Test has run, you can press the View a File function key to look at the output of the Translation Test. This helps you locate major problems in the grammar or in the translations.

For isolated problems, you can perform further testing on a subset of the generated sentence file or on a sentence file you have produced.

**Example 2** **13.4.2** Following is an example of the Translation Test results obtained from an edited version of a system-generated sentence file. The sentence file is the same as the one used in the previous example, except that it has been edited to contain errors that the Translation Test traps.

Edited Sentence File

```
(1)  #   Find wrkers

(2)  #   Find workers whose name is equal to < specific name >
     |   select * from workers where name = < expert text >

(3)  #   Find workers whose employee number is equal to
             < specific no. >
     |   select all from workers where employee
             number = < expert text >

(4)  #   Find workers whose name is equal to < specific name >
             and whose employee number is equal to
             < specific no. >

(5)  #   Find name of workers

(6)  #   Find employee number of workers

(7)  #   Find name and employee number

(8)  #   Find name of workers whose name is equal to
             < specific name >

(9)  #   Find name of workers whose employee number
             is equal to < specific no. >

(10) #   Find name of workers whose name is equal to
             < specific name > and employee number is
             equal to < specific no. >
```

Various errors are now in the input file: *workers* is misspelled (wrkers) in the first sentence; the input translation in sentence 3 is not the correct translation; sentences 7 and 8 are not complete and are, therefore, not executable. The Translation Test gives the results shown next for this file.

```
                    ***** begin Translation Test *****

sentence: Find wrkers
T0002 -   test could not assign grammatical category,
          skipping to next sentence
T0001 -   the input sentence was not executable, no
          translation generated

sentence: Find workers whose name is equal to
                    < specific name >
          input translation same as one generated:
          select * from workers where name = < expert text >

sentence: Find workers whose employee number is equal
                    to < specific no. >
T0004 -   input translation of:
          select all from workers where employee
                    number = < expert text >
          not same as one generated:
          select * from workers where employee
                    number = < expert text >

sentence: Find workers whose name is equal to
                    < specific name > and whose employee
                    number is equal to < specific no. >
          no input translation, translation generated:
          select * from workers where name = < expert text >
                    and employee number = < expert text >

sentence: Find name of workers
          no input translation, translation generated:
          select name from workers

sentence: Find employee number of workers
          no input translation, translation generated:
          select employee number from workers

sentence: Find name and employee number
T0001 -   the input sentence was not executable, no
          translation generated
```

```
sentence: Find name of workers whose is equal
                    to < specific name >
T0002 -   test could not assign grammatical category,
          skipping to next sentence
T0001 -   the input sentence was not executable, no
          translation generated

sentence: Find name of workers whose employee
                    number is equal to < specific no. >
          no input translation, translation generated:
          select name from workers where employee
                    number = < expert text >

sentence: Find name of workers whose name is equal
                    to < specific name > and employee number
                    is equal to < specific no. >

                    ----- Translation Test complete -----
```

The numbered statements (T0002 and so on) reflect errors the test encountered in processing the sentence. The unnumbered statements provide further information. By examining the output file, you can determine where the errors are—in the input sentence file, in the grammar, or in the lexicon. After making the appropriate corrections to the grammar and retesting it, rerun the Translation Test until it generates the proper target language translations for the interface.

# OTHER INTERFACE BUILDER OPTIONS 14

# THE NATURALLINK SESSIONER 15

## Introduction

**15.1** The NaturalLink Sessioner serves as the coordinator between the application program and the NaturalLink software during construction of a command sentence. To bring up the NLmenu screen, the application program calls the Sessioner with the interface filename and the saved sentences filename. The saved sentences filename must be supplied, even though it will not be used unless the user selects the Sentences option.

Upon entry, the Sessioner places the cursor on the first selectable item in the NLmenu screen and initializes the Parser. As the user builds and executes commands, the Sessioner coordinates the interaction among the Parser, the Translator, and Window Manager, passing control among these NaturalLink components and the application program.

## Command and Function Key Options

**15.2** Nine NaturalLink command options are available for any NLmenu screen. Each command option is associated with a command statement, such as Back Up, Start Over, Quit, and so forth, and a corresponding default function key. The associated function keys are **F3** through **F10** and the **ESC** key, which acts as the Quit key.

Once invoked, commands are under the direct control of the NaturalLink Sessioner. You cannot modify the operation of any of these NaturalLink command options; you can modify only the keys by which they are invoked.

The number of command options available to the user in the NLmenu screen depends on your application requirements. Only the Execute command is required by the NaturalLink Sessioner. The other eight commands are optional and can be either specified or left out, depending on the requirements of the application. You can also define your own command/function keys using **ALT**-function key combinations.

Following are the commands that you can specify for the Command window, with their default function keys.

- Execute/**F10** — Invokes the Translator; sends the NaturalLink command sentence and its translation to the application program.
- Customize/**F3** — Allows the user to make modifications to the phrases on the interface screen.
- Show/**F4** — Displays a pop-up window that shows the target-language translation string of the current command sentence.
- Edit/**F5** — Enables the user to edit the expert item(s) in the current command sentence.

- Sentences/**F6** — Invokes the saved-sentence processing function, which displays a pop-up window that contains a subset of one or more of the following commands (depending upon which commands in the subset are currently valid):

  - Save a command sentence
  - Recall a command sentence
  - Delete a command sentence

- Help/**F7** — Displays Help message(s). This option works in two ways, depending on whether it is invoked by the **F7** key or selected as an item:

  - Pressing the **F7** key causes display of the Help message(s) available for the item the cursor is on.
  - Placing the cursor on the Help/**F7** option in the command window and pressing the **ENTER** key causes display of a message that explains how to use the Help (**F7**) option.

- Back Up/**F8** — Erases the previous item or expert selection from the command sentence being built and returns the cursor to the previous active window.

- Start Over/**F9** — Erases the current command sentence and enables construction of a new NaturalLink command.

- Quit/**ESC** — Ends the NaturalLink session and returns to the application program.

- User-defined command/function keys — Special function and command key operations not supported internally by the NaturalLink Sessioner.

The following paragraphs describe the commands in detail.

## Execute Option

**15.2.1** Execute is the only NaturalLink command option required by the Sessioner. The default function key is **F10**. The NaturalLink Sessioner makes this command invisible in the NLmenu screen until the user has constructed a complete sentence (an executable NaturalLink command sentence). When a command sentence can be executed, the Sessioner displays Execute/**F10** in the command window. When Execute/**F10** is selected, the Sessioner terminates the command-building process, produces a translation string for the current command sentence, and returns that string, as well as the command sentence, to the application program for processing.

**Customize Option**

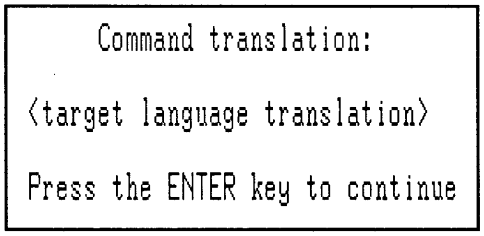
**15.2.2** This feature enables the user to make limited modifications to the interface screen. The default function key for this feature is **F3**. Two kinds of modifications are possible: editing phrases and adding synonyms. This function is also available as a routine that the application itself can call. See Chapter 16, Interface Customization Feature, for more information.

To allow use of the Customize option, you must explicitly link it in. Refer to Appendix C, High-Level Language Interface, for more information.

**Show Option**

**15.2.3** This command option enables the user to see the translation that will be returned to the application. The default function key for this feature is **F4**. Show is available only for an executable sentence; the Sessioner keeps this command window option invisible until the sentence becomes executable. When Show is selected, a pop-up window similar to the one shown in Figure 15-1 appears.

**Figure 15-1** **Show Translation Pop-Up Window**


```
      Command translation:

(target language translation)

Press the ENTER key to continue
```

**Edit Option**

**15.2.4** This command option enables the user to edit expert text fields in a NaturalLink command sentence either during construction of that sentence or after recalling a previously saved sentence. This option is of particular value when a number of command sentences have the same format but require different expert text. The default function key for this feature is **F5**.

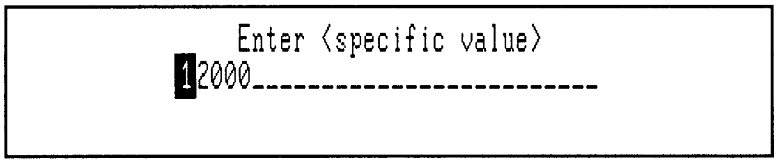
When Edit is invoked, the cursor is placed on the first expert text field in the current command sentence. The user selects the expert field to edit by moving the cursor to that field and pressing the **ENTER** key. A pop-up window appears in which the user can modify the expert text. This is the same pop-up window in which the user originally specified the expert text. (If it is a user-defined expert, the application routine NLXEXP is called.) Press the **ENTER** key to accept the edited expert text, update the command sentence, and return the cursor to the Command window. If there is more than one expert in the sentence, the cursor returns to the Results window. Pressing the **F10** key ends the Edit session.

Here is an example of the editing process in an NLmenu screen. Assume that the personnel department of a company frequently uses the following command sentence format to query its database:

> Find all workers whose salary is greater than 12000 and whose productivity output is less than 1000 units per month.

The values 12000 and 1000 are the expert items. If Edit is selected, these items are highlighted. Using the arrow keys, the user can then place the cursor on a value to be edited. If the value 12000 is selected for editing, a pop-up window similar to the one shown in Figure 15-2 appears. If this is a date expert, the old value does not appear.

**Figure 15-2** **Edit Expert Text**



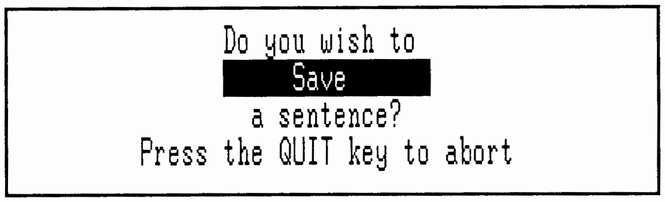
```
        Enter <specific value>
12000________________________
```

The new value entered in this pop-up window is checked for proper format (alphanumeric, minimum and maximum length, range, and so on), and the command sentence is updated to reflect the new expert value. The user can change only the value of the expert item(s), not the structure of the command sentence itself. During the construction of a NaturalLink command sentence, Edit interrupts the building process. The user can resume command sentence construction after editing. The NaturalLink Sessioner does not make the Edit option visible unless an expert item is available for editing in the current command sentence.

**Sentences Option**

**15.2.5** Invoking the Sentences option provides the user with access (through a pop-up window) to the Sessioner's three functions for manipulating saved sentences: Save, Recall, and Delete. The default function key for the Sentences option is **F6**. The availability of these functions is contingent upon the state of the command sentence files. If no command sentences have been saved, only the pop-up window shown in Figure 15-3 appears when Sentences is selected.

**Figure 15-3** **Saving a Command Sentence**



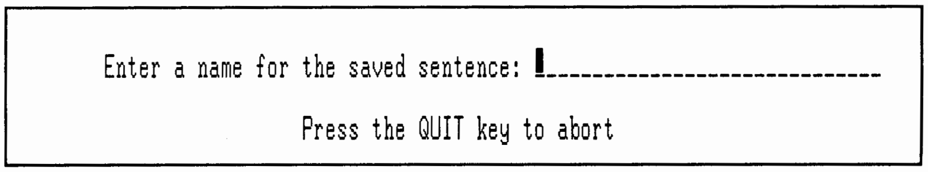
```
Do you wish to
     Save
  a sentence?
Press the QUIT key to abort
```

Do not allow use of the Sentences option unless you pass a saved sentences filename to the Sessioner in addition to the interface file name.

To allow use of the Sentences option, you must explicitly link it in. Refer to Appendix C, High-Level Language Interface, for more information.

If the user elects to save the current command sentence (this must be done before the command sentence is executed), another pop-up window (Figure 15-4) appears requesting the name under which the sentence is to be saved.

**Figure 15-4** **Naming the Sentence to Be Saved**

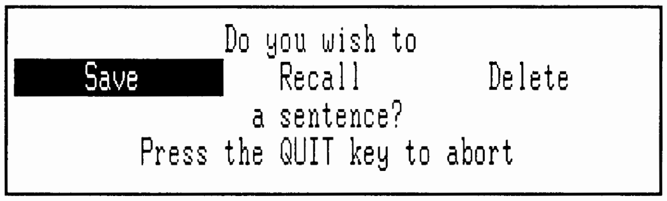


```
Enter a name for the saved sentence: ▌____________________________

                Press the QUIT key to abort
```

The name that identifies the saved sentence can be up to 30 characters long. To save the sentence, press the **ENTER** key after entering the name. If the name entered for a sentence matches an existing name, the message `This name has already been used` appears in the window just below the prompt line. The user is required to provide a unique name that is not null; the message `Please enter a name` appears if the **ENTER** key is pressed but the response field is empty. The maximum number of sentences that can be stored in a saved sentence file is 100. If the user attempts to save more than 100 sentences, an error message is displayed.

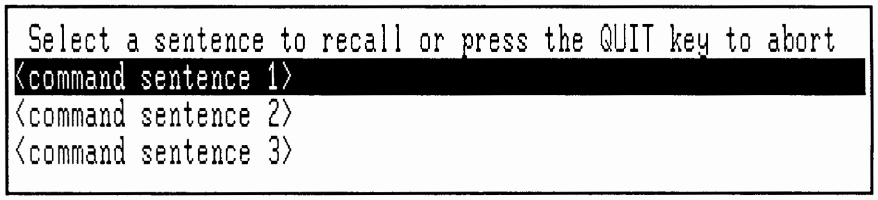
If a command sentence has been previously saved, the pop-up window shown in Figure 15-5 appears when the Sentences option is invoked.

**Figure 15-5** **Recalling or Deleting a Command Sentence**


```
            Do you wish to
   Save          Recall         Delete
              a sentence?
      Press the QUIT key to abort
```

If Recall or Delete is selected, a pop-up window similar to the one shown in Figure 15-6 appears.

**Figure 15-6** **Selecting a Sentence to Recall**


```
 Select a sentence to recall or press the QUIT key to abort
<command sentence 1>
<command sentence 2>
<command sentence 3>
```

To perform the Recall procedure, the user positions the cursor over the name of the sentence to be recalled (in this example, `<command sentence 1>`, `<command sentence 2>`, or `<command sentence 3>`) and presses the **ENTER** key. The Delete operation works the same way except that the prompt inside the pop-up window says `Select a sentence to delete or press the QUIT key to abort.`

**CAUTION: The Dynamic Lexical Items feature and the Synonyms portion of the Interface Customization feature (see Chapter 16, Interface Customization Feature) both modify the position of items in the interface screen by adding to and subtracting from the window item tables. DO NOT allow the user to save sentences from an interface for which either of these features is provided. Recalling a saved sentence when the interface screen items are not in the same positions WILL CAUSE A SYSTEM CRASH. The Edit Phrases portion of the Interface Customization feature does not modify item positions and can be used in conjunction with the Saved Sentences feature.**

**Help Option**

**15.2.6** This option, when invoked with a function key, displays Help message(s) specified for the item the cursor is on. The default function key is **F7**. Several Help messages can be chained to a single item. Each time the Help function key is pressed, the next Help message for an item is displayed until all have appeared. If item Help is not available, the Help message for the current window is displayed. When multiple Help messages are displayed, the user presses the Back Up function key (the default is the **F8** key) to return to the previous Help message. Pressing the **ENTER** key when a Help message is displayed returns the user to the NLmenu screen.

When invoked as an item selection (that is, the user places the cursor on the Help option in the Command window and presses the **ENTER** key), this option displays a Help message explaining how to use the Help function key.

**Back Up Option**

**15.2.7** This command option enables the user to go back one step in the current selection sequence. If a NaturalLink command is under construction, Back Up erases the previously selected item or expert from the current command sentence. When the user invokes Back Up while reviewing a series of Help messages, the last Help message displayed is removed from the screen and the previous Help message is displayed. The default function key for Back Up is **F8**.

**Start Over Option**

**15.2.8** This command option erases all selections for the current command sentence and enables the user to restart the NaturalLink command-building sequence. The default function key for Start Over is **F9**.

**Quit Option**

**15.2.9** This command option ends the NaturalLink command-building session and causes the program to exit the Sessioner. If the user invokes Quit before the command sentence under construction is executed, a null value is returned to the application program. The default function key for Quit is the **ESC** key.

The default function keys for Back Up, Start Over, and Quit do not work with user-defined experts unless you design them into the NLXEXP routine. Refer to paragraph 15.4, User-Defined Expert Routines.

## User-Defined Command/Function Keys

**15.2.10** You can define your own command and function keys. Using the NLXFUN routine, you can add item-selectable and function key-selectable options that are handled like the nine NaturalLink-supported command options. You can use **ALT**-key combinations to define function keys.

To define your own command option or function key, you first create an NLXFUN routine (see paragraph 15.4.4, User-Defined Function Key Handler). The Sessioner calls this routine when the user selects the item or presses the function key. (Unlike the NaturalLink-supported command and function key options, the function key for a user-defined function cannot be disabled through KBUILD.) After the routine has run, NLXFUN returns to the Sessioner.

A function key code must be associated with the defined command/function even if the key will not be used. Any extended hexadecimal key code will do (103H to 18FH) except those used by the Window Manager or the Sessioner. To make the user-defined command option item-selectable, include it as an item in the Command window and specify the associated hexadecimal key code as the item label text for that item. When the item is selected, the hexadecimal key code is sent to NLXFUN for processing.

## Invoking the Commands

**15.2.11** NaturalLink commands can be invoked by item selection, function key selection, or both. (The command Customize may be invoked in additional ways; see Chapter 16, Interface Customization Feature, for details.) In the NLmenu screen, the user executes an item-selectable command by placing the cursor on the command and pressing the **ENTER** key. To execute a function key–selectable item, the user simply presses the designated function key at the appropriate time. The default for the Sessioner is *both* item selection *and* function key selection (that is, the Sessioner accepts both command selection methods, unless you specify otherwise when you are building the interface).

If you do not want a particular command to be item selectable, it is recommended that you leave that command out of the NLmenu screen; do this by not including it as an item when you build the Command window or by setting the Unselectable Item attribute for that item to Yes when you build the window. (Remember, the Execute command must be specified and must be selectable.)

If you do not want a command to be function key selectable, set the default function key for that command to zero (0) using KBUILD, the function key change utility. Include the resulting function key object file in the link stream for your interface. (Refer to Chapter 10, Window Manager Input Devices, in the *NaturalLink Window Manager Reference Manual* for more information.)

**Identifying Command Options to the Sessioner**

**15.2.12** A special item label code is required for each command in the designated Command window. This code is set through the Edit Item Label option in Screen Builder. It enables the Sessioner to identify the command selected and to determine what action to perform. Table 15-1 shows each available command, along with its associated item label and default function key.

**Table 15-1** **NaturalLink Commands**

| ***Command*** | ***Default Function Key*** | ***Item Label Text*** |
|---|---|---|
| Execute | F10 | *X |
| Customize | F3 | *C |
| Show | F4 | *S |
| Edit | F5 | *E |
| Sentences | F6 | *M |
| Help | F7 | *H |
| Back Up | F8 | *R |
| Start Over | F9 | *O |
| Quit | ESC | *Q |
| User-defined command/ function keys | ALT-function key combinations | Extended hexadecimal key codes |

**NOTE:**

The extended hexadecimal key codes for the NaturalLink software are listed in Appendix C (Computer-Specific Information) in the *NaturalLink Window Manager Reference Manual.*

When setting the window format attributes for the Command window in Screen Builder, do not change the value for the Visible Item Labels option from the default value No to Yes. It may seem confusing to leave this value set to No when you have, in fact, designated item labels of *X, *Q, *H, and so forth, for the commands in the Command window. However, the value No for the Visible Item Labels option prevents the Sessioner from displaying the item labels in the NLmenu screen.

Function keys remain operable even if no corresponding item-selectable command is available in the designated Command window. To disable function keys, use KBUILD.

## NaturalLink Interface Calls

**15.3** The general call sequence for the NaturalLink Interface is as follows:

1. The application program calls the NaturalLink Window Manager initialization routine — WMINIT.
2. The application program loads the interface — NLLOAD.
3. The application program invokes the NaturalLink Sessioner (driver) — NLDRIV.
4. The application program unloads the NaturalLink Interface after a session has been completed — NLXUNL.
5. The application program calls the NaturalLink Window Manager keyboard reset routine — WMRSET.

The specific calls required for the languages supported by the NaturalLink Toolkit are described in Appendix C, High-Level Language Interface. For more information on parameter passing, status codes, and memory considerations, see the following appendixes in the *NaturalLink Window Manager Reference Manual*: Appendix D, C Interface; E, Pascal Interface; F, FORTRAN Interface; and G, Compiled BASIC Interface.

The Sessioner returns a status code for all calls described below. These status codes indicate whether an error or an abort call caused the Sessioner to terminate. It is the responsibility of the application program to check all returned status codes and to take the necessary action based on those codes. A return code of 0 (zero) indicates normal completion. For information about other error codes, refer to Appendix B, NaturalLink Error Codes, in this manual, and Appendix H, Window Manager Error Codes, in the *NaturalLink Window Manager Reference Manual.*

### Initialization and Termination

**15.3.1** When interacting with either the NaturalLink Sessioner or Window Manager, you must use the NaturalLink initialization and termination routines to ensure proper system behavior. To perform the necessary system initialization, call WMINIT before invoking any other NaturalLink routine. This routine sets up key mapping and cursor initialization. If you use Window Manager and/or the NaturalLink Sessioner, you need to make these calls only once in a program. When the application has made all calls to NaturalLink routines, call WMRSET to reset key mapping and cursor initialization. Refer to the *NaturalLink Window Manager Reference Manual* for more information about these routines. An explanation of the calls and parameters used for a NaturalLink interface follows.

The descriptions of calls in this section are generic. The order of the parameters may be different for each language.

## Loading the NaturalLink Interface

**15.3.2** Before invoking a NaturalLink interface, the application program must first preload the interface through the NLLOAD call sequence. This preloading procedure enables the software to handle the overhead of loading when the application is initially executed. Up to 10 interfaces can be loaded at one time (depending on interface size). This feature gives you the additional capability of splitting a complex application program into logical sections. The NaturalLink Sessioner assigns the numbers 0 through 9 (1 through 10 for FORTRAN) to the interfaces. Each interface is referred to by its assigned number until it is unloaded.

---

**CAUTION: While the NaturalLink software accepts up to 10 interfaces at a time, the actual number of interfaces you can load depends on the size of each interface and the amount of data space available. Fatal errors will occur if you exceed memory limitations. Consult the Memory Considerations section in the appendix for your application program language in the *NaturalLink Window Manager Reference Manual*.**

---

The NLLOAD call sequence is as follows:

| | |
|---|---|
| Name: | NATURALLINK INTERFACE LOADER |
| Call Sequence: | `NLLOAD`(*INTFILE*, *INTLEN*, *SAVEFILE*, *SAVELEN*, *INTNUM*) |
| Description: | Loads the specific interface file, if it is not already loaded. Returns *INTNUM* (a number 0–9, but 1–10 for FORTRAN), which is subsequently used to reference the loaded interface. |
| Parameters: | |
| *INTFILE*: | Input, a string. The interface file pathname. Must be a valid MS-DOS pathname. The file is loaded and used to determine the appearance of the screen, the available selections, and the translation of those selections while a command sentence is being built. |
| *INTLEN*: | Input, an integer. The length of the interface file pathname. |

| | |
|---|---|
| *SAVEFILE*: | Input, a string. The saved sentences file pathname. Must be a valid MS-DOS pathname. This is the actual pathname; it is used for the storage and retrieval of any sentences saved during the building of a command sentence. |
| *SAVELEN*: | Input, an integer. The length of the saved sentences file pathname. |
| *INTNUM*: | Output, an integer. The interface number returned by NLLOAD. Used to access the loaded interface in other calls. |

Argument *INTFILE* — Interface File Pathname. The interface file pathname identifies the interface file to be loaded. This string must be a valid MS-DOS pathname.

Argument *SAVEFILE* — Saved Sentences File Pathname. The saved sentences file pathname must also be a valid MS-DOS pathname. The file is used when the user invokes one of the commands for manipulating saved sentences: Save, Recall, or Delete. The Sessioner performs all saved sentence file maintenance (including creating the file). If you are not using the Sentences option, pass a null string for the *SAVEFILE* parameter. The file should not be modified or created by other programs, and *it is not valid for any interface except the one with which it is created.*

Argument *INTNUM* — Interface Number. The interface number is assigned by NLLOAD to identify the interface during later NLDRIV (NaturalLink Sessioner) and NLXUNL (Interface Unload) calls. Up to 10 interfaces can be loaded at any one time (within memory constraints).

## Invoking the NaturalLink Interface

**15.3.3** The application program invokes the NaturalLink interface with the NLDRIV call. This function returns an error status code as the function value.

The NLDRIV call is as follows:

| | |
|---|---|
| Name: | NATURALLINK DRIVER |
| Call Sequence: | `NLDRIV`(*INTNUM*, *XLATION*, *XLATLEN*, *RESTEXT*, *RESLEN*) |
| Description: | Controls the building of NaturalLink sentences using the interface number returned by NLLOAD. Also handles saved sentence file manipulation. |

Parameters:

| | |
|---|---|
| *INTNUM*: | Input, an integer. The interface to be used. The number was assigned by NLLOAD. |
| *XLATION*: | Output, a string. The buffer used to return the translation of the constructed sentence. Returned when the user invokes the Execute command. This buffer should be large enough to hold the longest possible translation string that can be generated by the current grammar. |
| *XLATLEN*: | Input, an integer. The length of the *XLATION* buffer that will contain the returned translation string. |
| *RESTEXT*: | Output, a string. The buffer used to return English text that appears in the Results window as the command sentence is generated. |
| *RESLEN*: | Input, an integer. The length of the *RESTEXT* buffer that will contain the returned English text. |

It is recommended that you place a null in the *RESTEXT* parameter if you do not want to save the actual text of the command sentence. This frees memory space.

Argument *INTNUM* — Interface Number. The interface number is assigned by NLLOAD to identify the interface during later NLDRIV (NaturalLink Sessioner) and NLXUNL (Interface Unload) calls.

Argument *XLATION* — Translation String. This is the string containing the Sessioner's translation of the sentence chosen by the user. The buffer length parameter *XLATLEN* should never be less than the length of the longest possible translation. The string is returned when the user invokes the Execute command. A null string is returned if the user aborts the Sessioner.

Argument *RESTEXT* — Results Window Text String. This string contains the text that was in the Results window when the Execute command was invoked. The buffer length parameter RESLEN should never be less than the longest possible Results window sentence. A null string is returned if the user aborts the Sessioner.

### Unloading the NaturalLink Interface

**15.3.4** The application can unload an interface with the NLXUNL call, thereby reclaiming the memory used by the interface. Unloading an interface does not change any other interface number or its order.

The NLXUNL call is as follows:

| | |
|---|---|
| Name: | NATURALLINK INTERFACE UNLOADER |
| Call Sequence: | `NLXUNL(`*INTNUM*`)` |
| Description: | Unloads and releases memory for the interface specified by *INTNUM.* |
| Parameters: | |
| *INTNUM*: | Input, an integer. The number of the interface to be unloaded. INTNUM was assigned to the interface by NLLOAD. |

Argument *INTNUM* — Interface Number. The interface number is assigned by NLLOAD to identify the interface during later NLDRIV (NaturalLink Sessioner) and NLXUNL (Interface Unload) calls.

## User-Defined Expert Routines

**15.4** The calling sequences for user-defined experts are explained in general here, and in language-specific form in Appendix C, High-Level Language Interface. Chapter 9 (Application Control of User Options) discusses the purposes of these routines and the process involved in generating a user-defined expert. The calling routines for user-defined experts are as follows:

- NLXEXP — Written by the application designer and called by the NaturalLink Sessioner when a user-defined expert is encountered.
- NLSETR — Called by the NLXEXP routine to pass the expert value and English results text for the value back to the NaturalLink Sessioner.
- NLXOLD — Called by the NLXEXP routine to retrieve the old value of the expert for editing purposes. Must be called before an NLSETR call is made.

Status
Returned:

| | |
|---|---|
| −1 | Indicates that an error was encountered in handling the user-defined expert. When NLXEXP returns to the NaturalLink Sessioner with a −1 value, the Sessioner returns to the application program with a Fatal Error Encountered status code. |
| 0 | Indicates success. |
| 1 | Has the same effect as the Sessioner Back Up option. When NLXEXP returns with a 1 value, the Sessioner removes the last choice from the command sentence being built. |
| 2 | Has the same effect as the Sessioner Start Over option. When NLXEXP returns with a 2 value, the Sessioner discards the command sentence being built and allows the user to start building a new one. |
| 3 | Has the same effect as the Sessioner Quit option. When NLXEXP returns with a 3 value, the Sessioner returns to the application with a User Aborted status code. |

## The NLXEXP Call

**15.4.1** The NaturalLink User-Defined Expert Handler (NLXEXP) routine is called by the NaturalLink Sessioner, not by the application program. The application programmer writes the NLXEXP routine that will be called. The following paragraphs explain the parameters the application's NLXEXP routine will receive and what return values will be expected. Because the NaturalLink Sessioner calls the NLXEXP routine, the application must include a routine of that name in order for the NaturalLink application to link successfully.

A library containing a dummy NLXEXP routine has been provided to resolve the external reference from the NaturalLink Sessioner if no application NLXEXP routine has been explicitly linked in.

Following is the general form of the NLXEXP call made by the NaturalLink Sessioner; this is the form in which the routine must be defined in the application program.

| | |
|---|---|
| Name: | NATURALLINK USER-DEFINED EXPERT HANDLER |
| Call Sequence: | `NLXEXP(`*EXPTYP*`)` |
| Description: | Called by the NaturalLink Sessioner when the user selects a user-defined expert item from the NLmenu screen. |
| Parameters: | |
| *EXPTYP*: | Input, an integer. The code assigned to the user-defined expert item during the lexicon-building process. This code should be used by the NLXEXP routine to determine which expert item was chosen and, therefore, how to elicit the expert information. |

## The NLSETR Call

**15.4.2** The Set NaturalLink User-Defined Expert Value and Results Text (NLSETR) call passes back to the Sessioner the application's expert value translation and the English text to be placed in the Results window. The form of the call follows.

| | |
|---|---|
| Name: | SET NATURALLINK USER-DEFINED EXPERT VALUE AND RESULTS TEXT |
| Call Sequence: | `NLSETR(`*TRANS*`,` *TRLEN*`,` *RESULT*`,` *RESLEN*`)` |
| Description: | Transfers the expert text and results text for a user-defined expert to the NaturalLink Sessioner; called by NLXEXP before it returns to the Sessioner. |
| Parameters: | |
| *TRANS*: | Input, a string. The translation of the expert to be substituted into the translation string returned to the application. |
| *TRLEN*: | Input, an integer. The length of the translation string. |
| *RESULT*: | Input, a string. The results text of the expert to be placed in the Results window. |
| *RESLEN*: | Input, an integer. Length of the results text. |

A nonzero value returned from NLSETR indicates that an error was encountered in transferring one or both of the text strings. A zero value indicates success. If the first byte of the input English string is null, or if the length is zero, the translation string is used for both the translation and the Results window text.

## The NLXOLD Call

**15.4.3** The Set NaturalLink User-Defined Expert Value (NLXOLD) call is used by the application's NLXEXP expert handler to retrieve the expert text (the *TRANS* parameter) from the previous NLSETR call. This allows the application to display the old value of the expert (using the Window Manager set string call WMSETS). See paragraph 15.2.4., Edit Option. This call must be made before an NLSETR call, because the old value is destroyed when the NLSETR call is made.

| | |
|---|---|
| Name: | GET NATURALLINK USER-DEFINED EXPERT VALUE. |
| Call Sequence: | `NLXOLD(`*OLDVAL*`,` *OLEN*`)` |
| Description: | Allows user-defined expert handling routines to edit old values of the expert. *OLDVAL's* first byte will be set to zero if this routine is called when the sessioner is not in Edit mode. |
| Parameters: | |
| *OLDVAL*: | Output, a string. The buffer to receive the old value of the expert's translation text. |
| *OLEN*: | Input, an integer. The size of the *OLDVAL* buffer. If the length of the translation is greater than *OLEN*, the translation text will be truncated and an error returned. |

## User-Defined Function Key Handler

**15.4.4** This routine handles function keys not recognized by the NaturalLink Sessioner or Window Manager. This allows the application to perform functions while a command sentence is being built. The NLXFUN routine is accessed two ways:

- The Sessioner calls NLXFUN when the user presses a function key that is not handled by the Sessioner or Window Manager or when a selected item in the Command window has a label that is not handled by the Sessioner.
- The application program calls NLXFUN if the Window Manager routine WMWRCV (Window Manager receive call) returns a key code that is not handled by Window Manager.

The NLXFUN routine is as follows:

| | |
|---|---|
| Name: | NATURALLINK DEFINED FUNCTION KEY HANDLER |
| Call Sequence: | `NLXFUN(`*KEYCOD*`)` |
| Description: | Called by the NaturalLink Sessioner when the Sessioner and the Window Manager do not recognize a key code received. |
| Parameters: | |
| *KEYCOD*: | Input, an integer. This is the key code of the unrecognized key. It is the extended key code plus 100H. (See Chapter 10, Window Manager Input Devices, in the *NaturalLink Window Manager Reference Manual.*) |
| Status Returned: | Same as status code returned for NLXEXP. (Refer to paragraph 15.4., User-Defined Expert Routines.) |

A library containing a dummy NLXFUN routine has been provided to resolve the external reference from the NaturalLink Sessioner if no application NLXFUN routine has been explicitly linked in.

# INTERFACE CUSTOMIZATION FEATURE 16

## Introduction

**16.1** The Interface Customization feature enables the user to modify the interface screen. Users may find the program easier to use if they can change the language to their own wording.

Users can modify the interface in two ways:

- Edit words or phrases
- Create synonyms of words or phrases

The user can select the Edit Phrases option to change a phrase on the screen. For example, a user who prefers the term *locate* to *find* can change the "Find" phrase to "Locate."

The user can select the Add Synonyms option to add another phrase that will invoke a particular function. For example, if one user likes the term *change* but another prefers *modify*, the first user can create a new item, "Change," as a synonym of "Modify." Thereafter, when the user selects either "Change" or "Modify," the same path through the interface will be taken.

You can set up the Interface Customization feature in several ways, according to your preference:

- As a Command window option like Saved Sentences (refer to Chapter 15, The NaturalLink Sessioner)
- As a function key (default **F3**) (again, refer to Chapter 15)
- As a different part of the application, called directly by the application (refer to paragraph 16.3.4, Calling NLXEDT Directly)
- As a separate utility (refer to paragraph 16.3.4, Calling NLXEDT Directly)

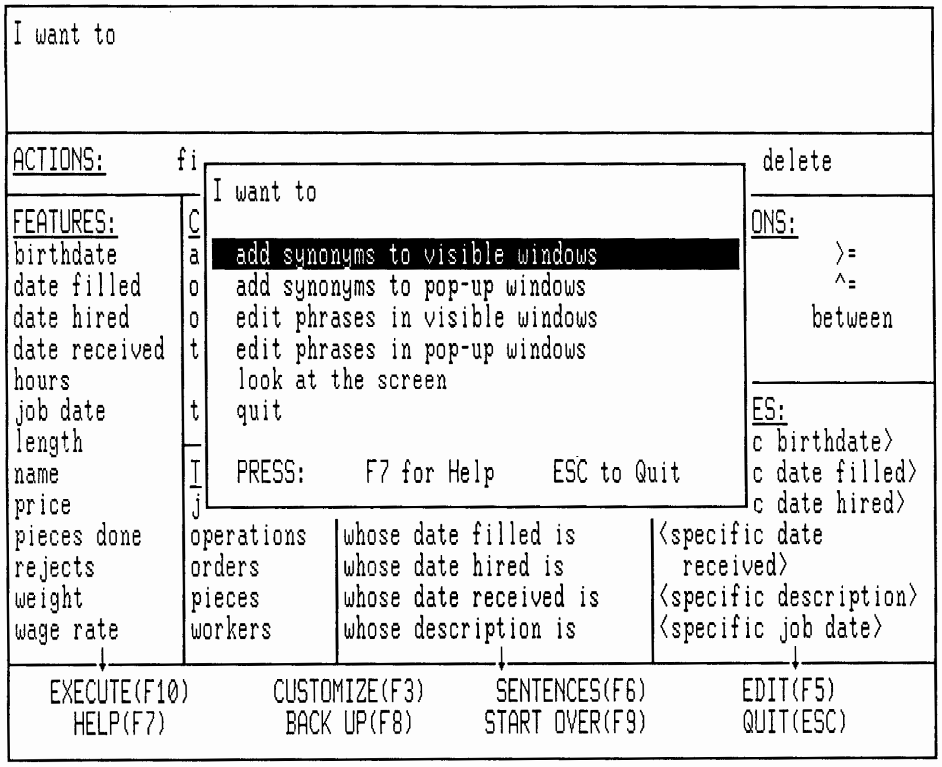
Because the Add Synonyms option adds another item to the interface screen and also adds a duplicate entry to the lexicon, the Add Synonyms option may not be desirable for large interfaces that have memory or speed problems. You can link in both options, Edit Phrases only, or neither. The Add Synonyms option cannot be linked in without the Edit Phrases option.

**CAUTION: The Dynamic Lexical Items feature and the Synonyms portion of the Interface Customization feature both modify the position of items in the interface screen by adding to and subtracting from the window item tables. DO NOT allow the user to save sentences from an interface for which either of these features is provided. Recalling a saved sentence when the interface screen items are not in the same positions WILL CAUSE A SYSTEM CRASH. The Edit Phrases portion of the Interface Customization feature does not modify item positions and can be used in conjunction with the Saved Sentences feature.**

## Instructions for the User

**16.2** If you decide to make both Interface Customization options available, you need to instruct the user on this feature. Figure 16-1 shows the Interface Customization menu (pop-up window) over an example NaturalLink menu.

**Figure 16-1** **Interface Customization Menu**

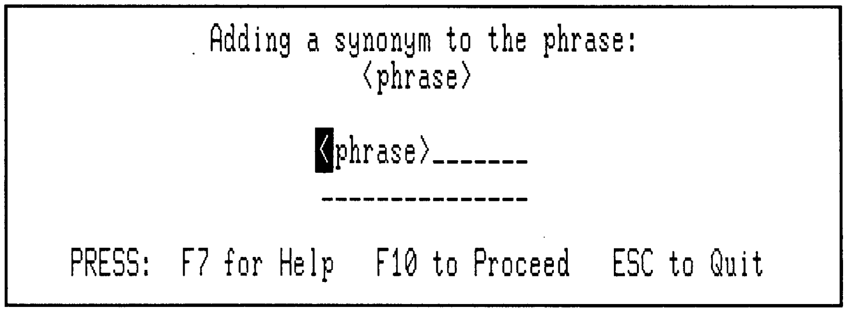


```
I want to

ACTIONS:      fi                                          delete

FEATURES:     C  I want to                                ONS:
birthdate     a   add synonyms to visible windows            >=
date filled   o   add synonyms to pop-up windows             ^=
date hired    o   edit phrases in visible windows          between
date received t   edit phrases in pop-up windows
hours             look at the screen
job date      t   quit                                    ES:
length                                                    c birthdate>
name          I   PRESS:    F7 for Help    ESC to Quit    c date filled>
price         j                                           c date hired>
pieces done   operations  whose date filled is    <specific date
rejects       orders      whose date hired is       received>
weight        pieces      whose date received is  <specific description>
wage rate     workers     whose description is    <specific job date>

   EXECUTE(F10)      CUSTOMIZE(F3)     SENTENCES(F6)      EDIT(F5)
     HELP(F7)         BACK UP(F8)     START OVER(F9)      QUIT(ESC)
```

**Add Synonyms to Visible Windows**

**16.2.1** If the user selects the first option, Add Synonyms to Visible Windows, the cursor is placed on the first item in the first visible window. The **CTRL-HOME**, **CTRL-PgUp**, **CTRL-PgDn**, **CTRL-Left Arrow,** and **CTRL-Right Arrow** keys move the cursor from window to window. After the user selects the item for which a synonym is desired, the pop-up window shown in Figure 16-2 appears, displaying that item.

**Figure 16-2** **Adding a Synonym to the Phrase**



```
                Adding a synonym to the phrase:
                           <phrase>

                        <phrase>_______
                        _______________

PRESS:  F7 for Help   F10 to Proceed   ESC to Quit
```

Two lines are available for completion in the display. The user starts typing over the word or phrase displayed and can continue until there is no more room. The **Down Arrow** or the **ENTER** key moves the cursor to the next line. This allows the addition of a synonym consisting of one or more words. When the new phrase is committed, it is added to the proper window, directly below the original phrase.

When all synonyms have been added to visible windows, pressing the Proceed key or the Quit key causes the cursor to return to the Interface Customization menu.

Note that users cannot create synonyms for items in the Command window, in the Results window, or in nonparse windows.

When a phrase is edited or a synonym added, the current value of the phrase being edited can take up no more than 23 lines in its window. A phrase longer than 23 lines will be truncated. For example, assume a phrase of 200 characters was somehow assigned to a window that is only 5 characters wide (can hold only 5 characters per line). If this long phrase were edited with the Edit Phrases option, only 115 characters (5 x 23) would appear in the window. The remaining 85 characters would be truncated. If you press the Quit key to abort the edit, no truncation will take place.

No synonyms can be created or phrases edited in user-defined windows that are part of the interface screen, whether or not the windows are parse windows. User-defined windows will not be displayed by the Interface Customization feature unless they were already displayed when the feature was invoked (as they are when the user presses the Customize function key). The user cannot move into these windows to edit or assign synonyms to any of the items in them.

## Add Synonyms to Pop-Up Windows

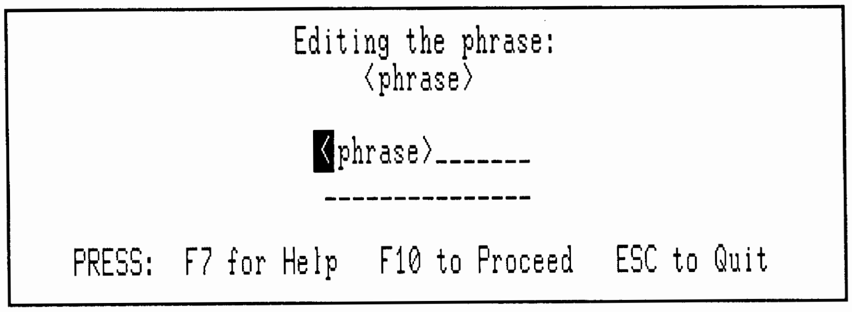
**16.2.2** This option allows the addition of synonyms to parse windows designated as pop-ups. If the Add Synonyms to Pop-Up Windows option is selected, the cursor is displayed on the first item of the first pop-up window. After the item is selected, a pop-up window displays the item, with space for typing the desired synonym.

When the user completes the changes desired for that item and commits the window, the cursor returns to the same pop-up parse window; pressing the Proceed key brings up the next pop-up window. All pop-up windows are presented according to their order in the interface screen file. After they have all been presented, pressing the Proceed key causes the cursor to return to the Interface Customization menu.

## Edit Phrases in Visible Windows

**16.2.3** This option allows editing of items in parse windows, the Command window, the Results window, and any other windows that appear in the interface screen. If this option is selected, the cursor is placed in the first visible window; when the word or phrase to be changed is selected, the pop-up window shown in Figure 16-3 appears.

**Figure 16-3** **Editing the Phrase Pop-Up Window**



```
                    Editing the phrase:
                         <phrase>

                     <phrase>_______
                     _______________

   PRESS:  F7 for Help   F10 to Proceed   ESC to Quit
```

This window displays the phrase to be edited, with the cursor on the first letter. The user changes the word or phrase as desired; after the window is committed, the new phrase replaces the old one. The **CTRL-HOME**, **CTRL-PgUp**, **CTRL-PgDn**, **CTRL-Left Arrow**, and **CTRL-Right Arrow** keys move the cursor from one visible window to another.

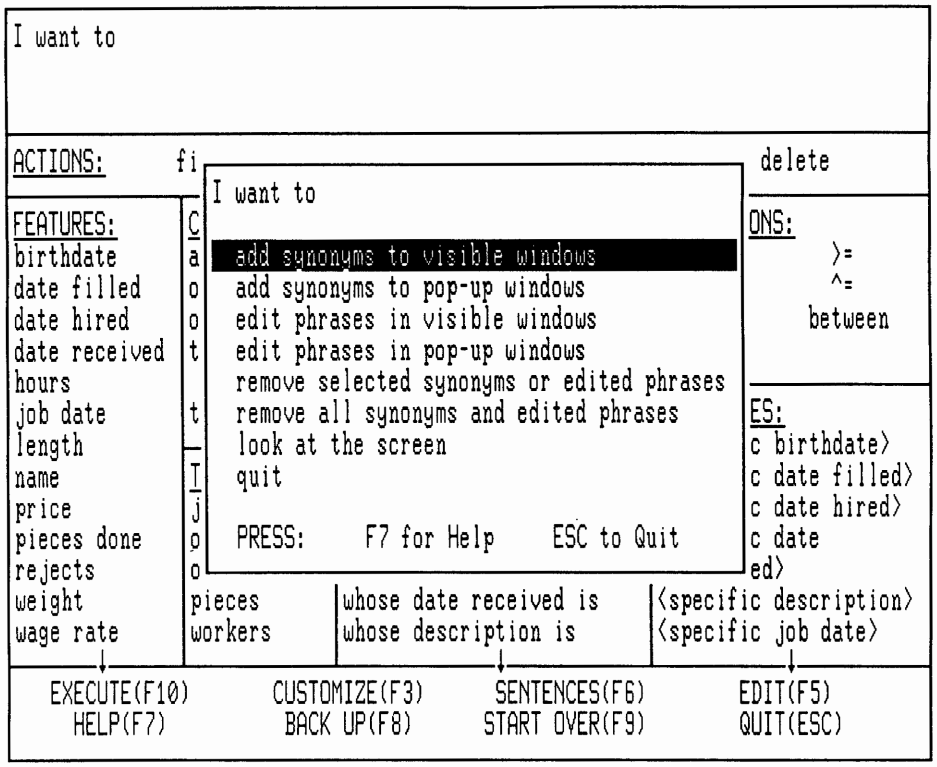
**Edit Phrases in Pop-Up Windows**

**16.2.4** With this option the user can edit words or phrases in the windows designated as pop-ups and edit the Results window label, if the label is being used as a flag that a sentence is executable (see Chapter 5, Using Screen Builder to Create an NLmenu). To enable editing of the Results window label, the Interface Customization feature treats it as a pop-up window. If the Edit Phrases in Pop-Up Windows option is selected, the cursor is placed first on the Results window label, if there is one. When the user selects the phrase to be modified, a pop-up window shows the phrase, with space for changes. After the desired changes are completed and the window committed, the Results window label disappears and each pop-up window is presented for editing in the order you designated in the screen file. If the user does not want to change the Results window label, pressing the Proceed key causes the first pop-up window to appear.

**Remove Selected Synonyms or Edited Phrases**

**16.2.5** This option is not available until at least one synonym is added or one word or phrase is edited; it is then added to the Interface Customization menu (Figure 16-4).

**Figure 16-4** **Expanded Interface Customization Menu**

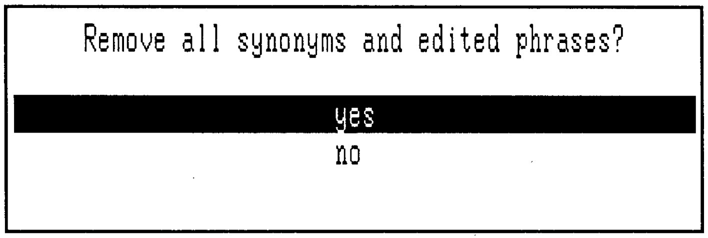

```
I want to

ACTIONS:      fi                                              delete
FEATURES:    C   I want to                                    ONS:
birthdate    a    add synonyms to visible windows                  >=
date filled  o    add synonyms to pop-up windows                   ^=
date hired   o    edit phrases in visible windows               between
date received t   edit phrases in pop-up windows
hours             remove selected synonyms or edited phrases
job date     t    remove all synonyms and edited phrases      ES:
length            look at the screen                          c birthdate>
name         I    quit                                        c date filled>
price        j                                                c date hired>
pieces done  o    PRESS:     F7 for Help     ESC to Quit      c date
rejects      o                                                ed>
weight       pieces      whose date received is    <specific description>
wage rate    workers     whose description is      <specific job date>

   EXECUTE(F10)       CUSTOMIZE(F3)      SENTENCES(F6)       EDIT(F5)
    HELP(F7)           BACK UP(F8)       START OVER(F9)      QUIT(ESC)
```

When the user selects the Remove Selected Synonyms or Edited Phrases option, all visible windows containing synonyms and edited phrases are displayed with all new items visible. To select the synonym or edited phrases to be removed, the user places the cursor on each one and presses **ENTER**. The **CTRL-HOME**, **CTRL-PgUp**, **CTRL-PgDn**, **CTRL-Left Arrow**, and **CTRL-Right Arrow** keys allow movement between visible windows. To access pop-up windows with newly created synonyms or edited phrases, the user presses the Proceed key. The Proceed key is also used to move between successive pop-up windows.

## Remove All Synonyms and Edited Phrases

**16.2.6** This option is not available until at least one synonym is added or one word or phrase is edited. When the option is selected, the pop-up window shown in Figure 16-5 appears, and the user must select Yes or No.

**Figure 16-5** **Remove All Synonyms and Edited Phrases Pop-Up Window**



```
Remove all synonyms and edited phrases?

                 yes
                 no
```

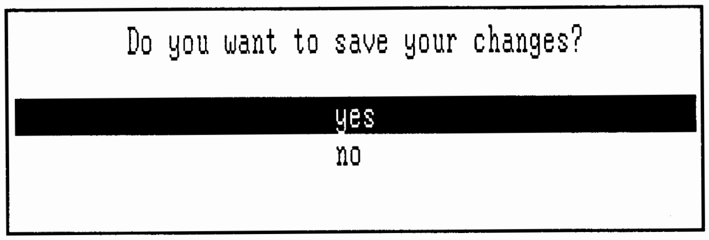
## Look at the Screen

**16.2.7** The user can view the entire screen at once by selecting this option. This may be desirable after the user has edited phrases or added synonyms. First all visible windows are shown, including the Results window label (which is treated as a pop-up only for editing phrases). Pressing the Proceed key causes each pop-up window to be displayed in succession. To return to the Interface Customization menu after viewing all windows, the user presses the Proceed key again.

## Quit

**16.2.8** The user can select this option at any time to leave the Interface Customization feature. When the option is selected, the pop-up window shown in Figure 16-6 appears. Making a selection from this window terminates the Interface Customization feature.

**Figure 16-6** **Quit Option Pop-Up Window**


Do you want to save your changes?

yes
no

## Other Information for the User

**16.2.9** Following is additional information for the user.

- Synonyms can be added only to items in the lexicon that are not expert items.
- Both uppercase and lowercase can be used for adding synonyms or editing words or phrases.
- Synonyms can be added to the newly created synonyms as well as to those already in the application program.
- When newly created synonyms or edited phrases are added to a window that has an alphabetical list, they may cause the list to be out of order.

## Information for Application Programmers

**16.3** Following is additional information for application programmers.

### Windows Used in Interface Customization

**16.3.1** You can modify the text shown in all the Interface Customization windows either by using the Phrase Builder utility (PBUILD) or by editing the global phrase file (WMSTRDEF). See Chapter 9 of the *NaturalLink Window Manager Reference Manual*, Internal Phrase Editing, for further information.

## Interface Customization File

**16.3.2** As modifications to the interface are made, the Sessioner builds a data structure that keeps track of the changes. When the user completes the modifications, quits the Interface Customization option, and specifies that the changes be saved, the interface file is updated with the modifications, and an Interface Customization file is written to the disk.

This Interface Customization file is linked to the interface file by name (< INTFILE> .NE$) and must not be deleted if the interface is ever to be restored to its original state. The Sessioner sets the Read-Only attribute for the Interface Customization file to discourage the user from deleting the file.

## Interface Customization Call

**16.3.3** The call for invoking the Interface Customization feature from an application is NLXEDT. All windows should be deleted before this call is made unless they are part of the interface screen. The NLXEDT call does the following:

1. Loads the interface (if it has not already been loaded)
2. Adds the interface screen windows (if they have not already been added)
3. Allows the user to make any modifications wished
4. Saves to disk Interface and Interface Customization files (if the user wishes)
5. Deletes interface screen windows (if they were added by NLXEDT)
6. Returns to the application

The interface is not unloaded by the NLXEDT call. A separate NLXUNL call must be made when the application wishes to unload the interface from memory.

The syntax of the NLXEDT call is:

STATUS = NLXEDT( *int_file_name*, *int_number*, *output_file_name*)

| | |
|---|---|
| *int_file_name* | Input; a character string; the name of the interface file to be modified; if null, it is assumed that the interface has already been loaded and *int_number* specifies the interface to modify. |
| *int_number* | Input/output; the address of an integer; the number of the interface loaded in memory; if the interface is loaded by the NLXEDT call, NLXEDT will set this parameter; otherwise, it is used as input to determine which interface to modify. |
| *output_file_name* | Input; a character string; the name of the file to save the modified interface to; this filename is also used to create the Interface Customization filename (<*output_file_name*>.NE$). |

If you implement the Interface Customization feature as a Command window option or as a function key in the sentence-building process, the Sessioner will make the call to the appropriate procedure when the user invokes the feature; the application does not need to make any additional program calls.

## Calling NLXEDT Directly

**16.3.4** When NLXEDT is called directly from an application, the application controls the point at which the interface information is unloaded from memory. A user may choose to customize the interface and make changes to it, and then abort the changes rather than save them to disk. In this case, NLXEDT returns to the application without saving the changes on disk but leaves in memory the modifications made to the interface information.

An error code of 99 is returned from the NLXEDT call if the user makes changes to the interface but chooses to abort rather than save the changes to disk. The application program must check for this code. If you have indicated that you want a clean copy in memory, the application should now make the NLXUNL and NLLOAD calls required to restore to its previous state the interface information in memory. However, since it is best for the Interface Customization feature to be invoked in a separate utility, the application does not need a clean copy of the interface in memory; the usual NLXUNL call before leaving the program is the only call that needs to be made.

## Letting the Sessioner Invoke the Feature

**16.3.5** The Interface Customization feature should never be invoked from the interface screen through a user-defined function key. This feature can be used only if no sentence is being built by the user; however, the application's user-defined function key routine has no way of knowing whether or not a sentence is being built when the user presses the function key. Because the Sessioner can control the availability of the Interface Customization feature based on the state of the sentence, have the Sessioner invoke the feature if you want to include it on the interface screen. Having the Sessioner invoke the feature, rather than calling NLXEDT directly, lessens overhead (in memory, for instance) and, because the Sessioner can control the feature's availability, avoids many problems.

When the Interface Customization feature is included as a function of the Sessioner (the Customize function key is not disabled and/or the Customize option is included in the Command window of the interface), the Sessioner handles reloading the interface information to restore the interface to its previous state. The interface retains its interface number to avoid confusion for the application.

The Customize option will not be available when the user is in the process of building a sentence.

# FINAL STEPS IN PREPARING THE NATURALLINK INTERFACE 17

## Packaging the NaturalLink Interface

**17.1** At this point, an application program linked with the NaturalLink software and Window Manager should be running with an interface you built with the NaturalLink Toolkit. These files must be put on the user's distribution disk:

- The application EXE program file
- Data or configuration files the application may require
- Screens the application uses through the Window Manager
- Help files associated with the screens used by the application program
- One or more interface files built by the Interface Builder utility
- Message files the application uses through the Window Manager
- The LIERRMSG.NM$ error definition file or equivalent (if used)

The user's distribution disk cannot be write-protected if the user will use that disk for execution and if any of the following conditions are met:

- You implement the Saved Sentences option.
- You implement the Interface Customization option.
- You implement the Dynamic Lexical Items option *and* choose to save the interface file to disk.

## Backing Up Files

**17.2** After you have built the interface file, linked the application, and completed the distribution disk, you may want to delete files on the Winchester drive that were used during interface development. Before deleting any of those files, back up the grammar, lexicon, expert, screen description, help, and message files used in creating the interface file. Save any other screens, message files, or code you used in creating the application for possible later use. An application often needs modification; also, if you want to create an interface for another application functionally similar to the one you have just worked on, modifying the existing interface is much quicker than starting from scratch.

Make backup copies of the following:

- Grammar file.
- Screen definition (NLmenu screen).
- Lexicon and expert files. These files were created interactively with the Interface Builder. The name assigned to the lexicon file is < *INTFILE*> .NL$. Expert information is stored in < *INTFILE*> .NP$.
- Interface file. While this file is easily recreated from the previous four files, it may be useful to retain a copy of it.
- Help file for the NLmenu screen.
- Any other screens and help files used with the application.

# EQUIPMENT REQUIREMENTS A

## Introduction

**A.1** The equipment requirements for using NaturalLink Toolkit differ, depending on whether an interface application is being developed or merely run.

## Equipment Necessary for Interface Development

**A.2** To use the NaturalLink Toolkit utilities for interface development, you need the following equipment:

- Texas Instruments Professional Computer (TIPC), Texas Instruments Portable Professional Computer (TIPPC), Texas Instruments BUSINESS-PRO™, IBM® PC, IBM PC/XT™, or IBM Personal Computer AT™,with at least two disk drives, one of which is a Winchester
- A minimum of 512K bytes RAM (where K equals 1024)
- MS-DOS (up to and including Version 3.xx)
- The MS-DOS link editor or a compatible link editor
- One of the following high-level languages:
  - Lattice C Compiler (up to and including Version 2.14)
  - MS-FORTRAN (up to and including Microsoft Version 3.2 or a version compatible with the machine you are using)
  - MS-Pascal (up to and including Microsoft Version 3.2 or a version compatible with the machine you are using)
  - MS-BASIC Compiler (TI Version 1.0 or a version compatible with the machine you are using)

BUSINESS-PRO is a trademark of Texas Instruments Incorporated.

IBM is a registered trademark and IBM PC/XT and IBM Personal Computer AT are trademarks of International Business Machines Corporation.

## Equipment Necessary for User Operation

**A.3** The user, who will be executing the NaturalLink application, will need the following equipment:

- TIPC, TIPPC, TI BUSINESS-PRO, Texas Instruments PRO-LITE™ Professional Computer, IBM PC, IBM PC/XT, or IBM Personal Computer AT.

- MS-DOS (up to and including Version 3.xx).

- A minimum of 128K bytes of RAM. Add the memory requirements of the application itself to determine how much RAM is required for the user's machine. Whether the user's machine needs a Winchester drive depends on the application software, not the presence of NaturalLink software.

PRO-LITE is a trademark of Texas Instruments Incorporated.

# NATURALLINK ERROR CODES B

## Introduction

**B.1** This appendix contains three lists of error messages provided by the NaturalLink software.

- Those reported in the grammar tests
- Those reported in the Translation Test
- Those reported by the NaturalLink run-time software

The error-reporting software uses the same template logic as that used in building translations.

## Grammar Tests

**B.2** The grammar tests return the following errors. The template slots are filled as follows:

@1 — Always replaced with a character

@2 — Always replaced with an integer representing a rule number

@3 — Always replaced with a column number

@4 — Always replaced with a location string

@5 — Always replaced with the first element in the grammar rule

### Grammar Tests Error Messages

**B.2.1** The messages reported in the grammar tests are as follows.

```
F0001 -- Rule #@2 (begins with the string @5) in the grammar
file has no terminating symbol.
```

User Action: Examine the rule specified, and add the missing "!".

```
F0002 -- Rule #@2 (begins with the string @5) in the grammar
file has no beginning symbol.
```

User Action: Examine the rule specified, and add the missing "@".

```
F0003 -- Rule #@2 (begins with the string @5) in the grammar
file has no translation list.
```

User Action: Examine the rule specified, and add the missing translation list.

```
F0004 -- Rule #@2 (begins with the string @5) in the grammar
file has too many elements.
```

User Action: A rule must have fewer than 11 rule elements. Examine the rule specified, and rewrite the rule as two or more rules.

```
F0005 -- Rule #@2 (begins with the string @5) in the grammar
file has too few elements. Each grammar rule must refer to a
mother and a left side element.
```

User Action: Examine the rule specified. This rule has no right side, which is an illegal construct in a context-free grammar. Review the grammar-writing tutorial, and correct the rule syntax.

```
F0006 -- Rule #@2 (begins with the string @5) in the grammar
file has unbalanced abbreviatory symbols in location @4 in
position @3.
```

User Action: Examine the rule specified for unbalanced abbreviatory symbols and proper nesting order. Review the discussion on abbreviatory symbols, and correct the rule.

```
F0007 -- Rule #@2 (begins with the string @5) in the grammar
file has a bad character @1 in location @4 in position @3.
```

User Action: Rule specified has an illegal character. Review the grammar rule format, and correct the rule.

```
F0008 -- Rule #@2 (begins with the string @5) in the grammar
file has a complex left corner.
```

User Action: The rule specified has an abbreviatory symbol as the first element of the right side. Review the grammar rule syntax, and correct the rule.

```
F0009 -- Rule #@2 (begins with the string @5) in the grammar
file has parentheses immediately inside braces in position @3.
```

User Action: The rule specified has a construct that results in a meaningless rule expansion. Review the discussion on abbreviatory elements, and correct the rule syntax.

```
F0010 -- Rule #@2 (begins with the string @5) in the grammar
file has an invalid translation list number in position @3.
```

User Action: The rule specified has a translation list that contains an integer greater than the number of elements in the rule or has a sequence of integers that does not reflect a possible rule expansion. Review the discussion of translation list syntax, and correct the rule.

```
F0011 -- Rule #@2 (begins with the string @5) in the grammar
file has no semantic substitution list in its translation
information.
```

User Action: The translation list of the rule specified does not contain the required two sequences of integers. Review the discussion of translation list syntax, and correct the rule.

```
F0012 -- Rule #@2 (begins with the string @5) the grammar
file has too many levels of nesting at position @3.
```

User Action: The rule specified contains more open abbreviatory symbols "(", "[", or "{" than the allowed number of rule elements per rule. The current limit is 10. To have a meaningful rule expansion, at least one rule element must be present for each open abbreviatory symbol. Therefore, more than 10 unclosed abbreviations are not allowed. Correct the rule.

```
F0013 -- Rule #@2 (begins with the string @5) in the grammar
file has braces immediately inside braces in position @3.
```

User Action: The rule specified has a wasteful construct. Simplify the rule as discussed in the grammar syntax section.

```
F0014 -- Rule #@2 (begins with the string @5) in the grammar
file has a missing @4 in position @3.
```

User Action: The rule specified has failed the syntax check because part of the rule, such as the translation list, is missing. Review the grammar syntax, and correct the rule.

```
F0015 -- Rule #@2 (begins with the string @5) in the grammar
file has only optional elements inside abbreviatory symbols
in position @3.
```

User Action: The rule specified has a meaningless rule expansion. Review the abbreviatory symbol discussion, and correct the rule.

```
F0016 -- Rule #@2 (begins with the string @5) in the grammar
file has a missing translation list delimiter (;).
```

User Action: The rule specified has failed the syntax check because of the missing ";". Review the grammar syntax discussion, and correct the rule.

```
F0017 -- Rule #@2 (begins with the string @5) in the grammar
file has parentheses inside the syntactic path list in its
translation information.
```

User Action: The first sequence of integers in the translation list of the rule specified contains nested parentheses. Nesting is not allowed in this sequence. Review the discussion of translation list syntax, and correct the rule.

```
F0021 -- Rule #@2 (begins with the string @5) in the grammar
file is missing the first open parenthesis in its translation
information.
```

User Action: The translation list for the rule specified is missing the first parenthesis. Review the translation list syntax, and correct the rule.

```
F0022 -- Rule #@2 (begins with the string @5) in the grammar
file is missing a closing parenthesis in its translation
information.
```

User Action: The translation list for the rule specified is missing the last parenthesis. Review the translation list syntax, and correct the rule.

```
F0023 -- Rule #@2 (begins with the string @5) in the grammar
file has translation list numbers in the syntactic path list
that are not in ascending order.
```

User Action: The first sequence of integers for a rule expansion is incorrectly entered in the translation list for the rule specified. This first sequence specifies the path taken through the rule and, therefore, must be in ascending order. Review the translation list discussion, and correct the rule.

```
F0026 -- Rule #@2 (begins with the string @5) in the grammar
file has a bad semantic substitution list. One of the numbers
in the list either doesn't exist in the syntactic path list
or is duplicated in the semantic substitution list.
```

User Action: The second sequence of integers for a rule expansion is incorrectly entered in the translation list for the rule specified. This second sequence must be in agreement with the first set of integers; no integer can appear that does not appear in the first set. Review the translation list discussion, and correct the rule.

## Translation Test

**B.3** The Translation Test returns the following errors. Strings are inserted in the template as follows:

@1 — Always replaced with input translation

@2 — Always replaced with generated translation

### Translation Test Error Messages

**B.3.1** The messages reported in the Translation Test are as follows.

```
T0001 -- The input sentence was not executable; no
translation generated.
```

User Action: The Translation Test completed parsing the sentence and determined it was not executable. Reexamine the grammar to ensure that the statement is complete, and correct the sentence.

```
T0002 -- Test could not assign grammatical category, skipping
to next sentence.
```

User Action: The Translation Test encountered a category during sentence parsing that was not in the set of categories the Parser expected next. Reexamine the sentence to ensure that it conforms to the grammar, and correct the sentence.

```
T0003 -- Test found completed sentence before end of
complete sentence.
```

User Action: During the parse of the input sentence, the Translation Test found a complete, noncontinuable sentence before the whole input sentence was processed. Reexamine the sentence to ensure that it conforms to the grammar, and correct the sentence.

```
T0004 -- Input translation of @1 not same as generated
translation: @2.
```

User Action: The translation expected by the user is not the same as the one generated by the Translation Test. Check the input translation and grammar and lexicon translation information. Make corrections as needed.

## NaturalLink Run-Time Software

**B.4** Errors encountered in the NaturalLink run-time software are returned to the calling application program. Refer to Appendix H, Window Manager Error Codes, *Window Manager Reference Manual*, for error codes returned from the Window Manager. The following codes are returned. Some of these codes are simply informative; these are identified as FYI (For Your Information). Others are nonrecoverable errors; these are identified as FATAL.

### Run-Time Error Codes

**B.4.1** The run-time error codes are as follows.

**26—Unrecognized function (FATAL)**

The NaturalLink Parser received an unrecognized request code. The executable module has been irreparably damaged. Recopy your executable module from the distribution disk and reexecute the program.

**27—Bad translation node (FATAL)**

The NaturalLink Translator encountered an error while constructing translations. The interface file has been irreparably damaged. Have your software analyst generate a new interface file for this application.

**28—Bad data in interface file**

The interface file @1 has been damaged. Make a new copy of it, or, if this was your only copy, regenerate the interface file. If the problem persists, check the status of your disk to be sure that it has not been damaged.

**30—Not an interface file**

The file @1 is not an interface file. The file may have been damaged, the wrong pathname may have been specified, or a different type of file may have been copied over the interface file desired.

### 62—User aborted (FYI)

The user aborted the NaturalLink Sessioner using the **ABORT** key, a user-defined expert, or a user-defined function key.

### 63—Sessioner error (FATAL)

The NaturalLink Sessioner has encountered an error. Check for damage to your program and try again.

### 64—Create saved sentences file error (FATAL)

The NaturalLink Sessioner was unable to create the saved sentences file. Check your disk for directory and disk space and for damage.

### 65—Damaged saved sentences file

The saved sentences file has been damaged. Possibly a file of a different type was copied over the saved sentences file. If you have another copy of this file, replace it. Otherwise, delete the file and resave any sentences that were in the file.

### 66—Topic directory full (FYI)

The saved sentences file is full. Only 100 questions can be saved in one saved sentences file.

### 70—Saved sentences read error

An error was encountered in trying to read the saved sentences file. The file has probably been damaged. Get a new copy of it if you can; otherwise, delete the file and resave your queries.

### 71—Not a saved sentences file

The file designated as a saved sentences file is invalid. There are two possible causes: 1) the wrong file pathname was specified 2) the file was associated with a different interface, and 3) a file of a different type was copied over the saved sentences file desired.

### 72—Invalid variable text separator error

An invalid variable text separator was received by the DISMSG call. Valid ASCII values for variable separators are 0 through 127.

## 73—Interface loader full (FYI)

Ten interfaces have already been loaded. To load another interface, unload a current interface using the interface unload routine.

## 74—Invalid interface number

An invalid interface number was given to either the Sessioner (driver) or the interface unloader. Make sure that the number specified for the routine is valid.

## 75—Interface file not found

The interface loader cannot find a file designated as the interface file. Check the file pathname for the interface file.

## −76—Interface file already loaded (FYI)

The interface file given to the interface loader has already been loaded into memory.

## 77—Invalid string pointer passed

A string pointer passed into this routine points to an invalid or otherwise inappropriate string. Check the string type and length.

## 78—NLXFUN error

An error code was returned from the user-defined function key routine. Handle this error as appropriate for your routine.

## 79—NLXEXP error

An error code was returned from the user-defined expert routine. Handle this error as appropriate for your routine.

## 80—Invalid saved DTS information file error

The saved Dynamic Lexical Items Information file associated with this interface is not a valid saved information file. Another file may have been copied over this file, or the file may have been damaged. Check the file, and try again.

### 81—Invalid version number error

The version number in the Dynamic Lexical Items Information file specified to NLXCRE is invalid. Check the manual for the correct version number, and correct the file.

### 82—Invalid command marker error

The Dynamic Lexical Items Information file specified has an invalid command marker. Check the manual, and correct the file.

### 83—Missing window label error

The Dynamic Lexical Items Information file is missing a window label. Check the manual, and correct the file.

### 84—Missing item error

The Dynamic Lexical Items Information file is missing an item. Check the manual, and correct the file.

### 85—Invalid command line error

The Dynamic Lexical Items Information file has an invalid command line. Check the manual, and correct the file.

### 86—Invalid number of entries line error

The Dynamic Lexical Items Information file has an invalid number of entries line. Check the manual, and correct the file.

### 87—Invalid array pointer error

One of the information array pointers passed to NLXSND is invalid. Check the manual, and correct the error.

### 88—Invalid number of entries error

The number of dynamic lexical entries to create that was specified either to NLXSND or in the Dynamic Lexical Items Information file is invalid. Check the manual for the correct syntax, and correct the call or the file.

### 89—Save without output file error

The Save or Unload option was specified to NLXCRE when an output file was not specified. Correct the error, and retry the operation.

### 90—No information specified error

A call was made to NLXCRE with no information about what to do. Either a call must tell NLXSND what entries to create or remove, or an information file must be specified to NLXCRE.

### 91—Invalid option error

An invalid option was specified to NLXSND or NLXCRE. Check the manual, and correct the error.

### -92—Invalid window label warning

NLXCRE was unable to create an entry because of an invalid window label. All other entries were created as specified. Check the Dynamic Lexical Items information specified, and correct the error.

### -93—Invalid DTS code warning

NLXCRE was unable to create an entry because of an invalid DTS code. All other entries were created as specified. Check the Dynamic Lexical Items information specified, and correct the error.

### 94—Cannot create error

Information was specified for dynamic lexical items to be created, but the Restore or Delete option was specified. Lexical items can be created only when the Add or Replace option is specified.

### 95—Line too long error

A line over 80 characters long was encountered in the Dynamic Lexical Items Information file. The maximum line length for this file is 80 characters. Correct the error, and try again.

## 96—Error on file attribute error

An error was encountered in trying to set or clear the Read-Only attribute on either the Saved Dynamic Lexical Items Information file or the Saved Editing Phrases Information file. Check for problems with your disk.

## 97—Conflicting options error

Conflicting options were specified to NLXSND or in the Dynamic Lexical Items Information file for the same dynamic terminal symbol (DTS) code. Check the manual, and correct the error.

## -98—Restore with no modifications warning

The RESTORE option was specified to NLXCRE when no Dynamic Lexical Items modifications file was found for this interface. Either no modifications have been made to this interface or the modifications file (with the .ND$ extension) was not copied into the same directory as the interface file. The RESTORE option has been ignored.

## 99—User aborted edit error

The user quit the Interface Customization session and chose not to save the changes that were made to the interface. To remove the changes from the copy of the interface in memory, unload the interface from memory and reload it. The interface file on disk has not been modified.

## 100—No editable windows error

None of the windows in this interface can be edited. See Chapter 16, Interface Customization Feature, for a discussion of this feature.

## – 101—Memory flushed warning

The WMFLSH call erased NaturalLink's memory. All screens and interfaces have been unloaded from memory. You must reload anything you need to use.

## 102—Invalid operation on NLmenu screen

An invalid operation was attempted on the Command window, the Results window, or a parse window in the NLmenu screen. The operation was probably an attempt to change a window attribute or the item table of one of these windows. The operation attempted cannot be performed on the Command window, the Results window, or any parse window in the NLmenu screen.

# HIGH-LEVEL LANGUAGE INTERFACE C

## Introduction

**C.1** This appendix contains information on calling and linking the NaturalLink software for the Lattice C, MS-FORTRAN, MS-Pascal, and MS-BASIC compilers. The calling sequences for the following routines are described.

- Sessioner routines NLLOAD, NLDRIV, NLXUNL, NLSETR, and NLXOLD (discussed in Chapter 15, The NaturalLink Sessioner)
- Dynamic lexical items routines NLXSND and NLXCRE (discussed in Chapter 9, Application Control of User Options)
- Interface customization routine NLXEDT (discussed in Chapter 16, Interface Customization Feature).

Also described are the application expert and key function routines (NLXEXP and NLXFUN) called by NaturalLink (discussed in Chapters 9 and 15).

## Calling and Linking With the NaturalLink Software

**C.2** An explanation of the characteristics of the NaturalLink High-Level Language Interface (HLL), including the way strings are passed for each of the languages supported, can be found in Appendices D, E, F, and G of the *NaturalLink Window Manager Reference Manual.*

One particularly important characteristic of the HLL for an application program using the NaturalLink Sessioner (driver) is the manner in which memory is assigned to the NaturalLink software. The amount of memory assigned is determined by a module that is linked with the software and the application. The default amount of memory is 2K bytes of stack and 32K bytes of heap. This is usually sufficient for an application using the NaturalLink Window Manager. However, the NaturalLink Sessioner and Parser may require more than 32K bytes of heap. If a get-memory error is reported while you are building a query, you should increase the heap space allocated to the NaturalLink software to a maximum of 55K and try again.

You can increase or decrease the amount of memory by modifying the memory allocation module. The name of the file you must modify, as well as other information about memory, can be found in the *NaturalLink Window Manager Reference Manual.* Refer to the Memory Considerations paragraph in the appendix for your specific language.

## Callable Routines — Lattice C Interface

**C.3** The following paragraphs describe the specific calling sequence and linking information for Lattice C.

### Calling Syntax

**C.3.1** Following is an example of the call for the NaturalLink Sessioner from Lattice C:

```
char *topic = "database.int" ;
char *userqs = "saves.svs" ;
char xlation[128] ;
char restext[256] ;
int stat ;
int intnum ;

stat = nlload(topic, 12, userqs, 9, &intnum) ;

stat = nldriv(intnum, xlation, 128, restext, 256) ;

stat = nlxunl(intnum) ;
```

It is recommended that you specify a null pointer for the `restext` parameter and a length of 0 (zero) if you do not want the actual text of the command sentence. This will free memory space that would be used to store this string.

**Example:**

```
  stat = nldriv(intnum, xlation, 128, null, 0) ;
```

The NLXEXP routine should be defined as follows:

```
  int nlxexp(exptyp)
  int exptyp ;
```

The NLXEXP support calls should be coded as follows:

```
int stat;
char xlation[80] ;
char restext[80] ;

stat = nlxold(xlation, 80) ;

stat = nlsetr(xlation, 80, restext, 80) ;
```

The NLXFUN routine should be defined as follows:

```
  int nlxfun(keycod)
  int keycod ;
```

A library containing dummy routines for NLXEXP and NLXFUN has been provided. If you do *not* use these routines, the dummy routines resolve the NaturalLink references to them.

Following is an example of Lattice C calls for the dynamic lexical items feature:

```
char *topic = "database.int";
char *outfil = "newdata.int";
int stat;
int intnum;
char *win_arr[3];
char *item_arr[3];
char *tran_arr[3];

win_arr[0] = "FIELDS:";
item_arr[0] = "employee number";
tran_arr[0] = "EMPNUM";
win_arr[1] = "FIELDS:";
item_arr[1] = "address";
tran_arr[1] = "EMPADD";
win_arr[2] = "FIELDS:";
item_arr[2] = "name";
tran_arr[2] = "EMPNAM";

stat = nlxsnd(14,2,3,win_arr,item_arr,tran_arr);

stat = nlxcre(topic,12,&intnum,NULL,0,outfil,11,0,0);
```

Following is an example of the Lattice C call for the Interface Customization feature:

```
char *topic = "database.int" ;
char *outfil = "newdata.int" ;
int  stat ;
int  intnum ;

stat = nlxedt(topic, 12, &intnum, outfil, 11);
```

**Link Stream** **C.3.2** To link a Lattice C application program with the NaturalLink Sessioner, you must include both the NaturalLink Window Manager and the NaturalLink Sessioner software. Following is an example of a link control file for the MS-DOS Linkage Editor. The question mark (?) appearing in the module names should be replaced with an S (small model), a P (large program model), a D (large data model), or an L (large program and data model), depending on the Lattice C compiler used for the application software.

| | |
|---|---|
| LILC?SH + | Memory organization module **must** occur first |
| C? + | C program entry/exit module |
| mymain + | The C application's main program |
| ... + | Any application subroutines used |
| ... + | Any NaturalLink called application routines used |
| ... | Modules for optional NaturalLink features |
| mymain /M | The .EXE file |
| mymain.MAP | The .MAP file |
| mylib + | Any application libraries used |
| LILC? + | Library of High-Level Language Interface routines to NaturalLink |
| NL + | NaturalLink Sessioner object library |
| WM + | Window Manager object library |
| APP??? + | Dummy NaturalLink called application routines |
| LC? | C run-time library |

If you are using a version of the Lattice C compiler earlier than 2.00, you must use C.OBJ instead of C?.OBJ and LC.LIB instead of LC?.LIB in your link stream.

The APP??? library contains dummy routines for all application routines called by NaturalLink. These include NLXEXP, NLXFUN, APPVAL, APPRCV, and so on. If you do not use all the features provided by these routines, the dummy modules in this library will resolve the NaturalLink references to them. If you have your own versions of these routines, they must be explicitly linked in following your main program. If you create libraries for any of your application routines, these libraries **must** be included before any NaturalLink libraries in the link stream. Replace the question marks in the APP??? library as follows:

- LCS for S model Lattice C
- LCD for D model Lattice C
- OTH for P or L model Lattice C

If you choose to implement the Saved Sentences feature, you must include the module NLSAVQRY.OBJ in your link stream.

If you implement the Dynamic Lexical Items feature, include the modules NLDTLIB.OBJ and NLISLIB.OBJ in the link stream.

If you plan to include the Interface Customization feature, include the modules NLEDLIB.OBJ, NLSYLIB.OBJ, and NLISLIB.OBJ. If you do not wish to support Adding Synonyms but only the Editing Phrases function of the Interface Customization feature, do not include the NLSYLIB.OBJ module. If you do not wish to support the Interface Customization feature at all, do not include any of these three modules.

NLISLIB.OBJ is used for both the Dynamic Lexical Items feature and the Interface Customization feature. You must include this module if you use either of these features.

The order of this link stream is extremely important. Do not alter the order of the modules and libraries shown in the example when linking your application. All modules in this link stream must be included, regardless of the NaturalLink features implemented. Refer to Appendix D, C Interface, in the *Window Manager Reference Manual*, for more information on linking.

## Callable Routines — Pascal Interface

**C.4** The following paragraphs describe the specific calling sequence and linking information for MS-Pascal.

### Calling Syntax

**C.4.1** The routine declarations for the NL routines are located in the file called NLEXTRNS.PAS. You must include them in the application prior to using calls to the routines.

An example of a call to the NaturalLink Sessioner from Pascal follows.

```
VAR
infile :  PACKED ARRAY [1..12] OF CHAR ;
sqfile :  PACKED ARRAY [1..9] OF CHAR ;
xlation :   PACKED ARRAY [1..128] OF CHAR ;
restext :  PACKED ARRAY [1..256] OF CHAR ;
stat :    INTEGER ;
intnum :  INTEGER ;

infile := 'database.int' ;
sqfile := 'saves.svs' ;

stat := nlload(infile[1], 12, sqfile[1], 9, intnum) ;

stat := nldriv(intnum, xlation[1], 128, restext[1], 256) ;

stat := nlxunl(intnum) ;
```

It is recommended that you place an array of size 1 in the `restext` parameter and a length of 0 (zero) if you do not want the actual text of the command sentence. This will free memory space that would be used to store this string.

The NLXEXP routine should be defined as follows:

```
  function nlxexp(exptyp : integer) : integer [public];
```

Calls to the NLXEXP support routines appear as follows:

```
VAR
xlation :  PACKED ARRAY[1..80] OF CHAR ;
restext : PACKED ARRAY[1..80] OF CHAR ;
stat:    INTEGER ;

stat := nlxold(xlation[1], 80) ;

stat := nlsetr(xlation[1], 80, restext[1], 80) ;
```

The NLXFUN routine should be defined as follows:

```
  function nlxfun(keycod : integer) : integer [public];
```

A library containing dummy routines for NLXEXP and NLXFUN has been provided. If you do *not* use these routines, the dummy routines resolve NaturalLink references to them.

Following is an example of the MS-Pascal call for the Dynamic Lexical Items feature:

```
VAR
infile : PACKED ARRAY [1..12] OF CHAR ;
otfile : PACKED ARRAY [1..11] OF CHAR ;
iffile : PACKED ARRAY [1..2] OF CHAR ;
stat : integer ;
intnum : integer ;
window : PACKED ARRAY [1..7] OF CHAR ;
item : PACKED ARRAY [1..15] OF CHAR ;
tran : PACKED ARRAY [1..6] OF CHAR ;

window := 'FIELDS:' ;
item := 'employee number' ;
tran := 'EMPNUM' ;
stat := nlxsnd(14,2,window[1], 7, item[1], 15, tran[1], 6) ;
```

```
item := 'address' ;
tran := 'EMPADD' ;
stat := nlxsnd(14,2,window[1], 7, item[1], 7, tran[1], 6) ;

item := 'name' ;
tran := 'EMPNAM' ;
stat := nlxsnd(14,2,window[1], 7, item[1], 4, tran[1], 6) ;

infile := 'database.int' ;
otfile := 'newdata.int' ;

stat = nlxcre(infile[1],12,intnum,iffile[1],0,
              otfile[1],11,0,0);
```

---

**NOTE:** The NLXSND call in Pascal differs from the call in C because an array of character pointers cannot be passed in Pascal. A separate NLXSND call must be made for each dynamic lexical item to be created.

---

Following is an example of the MS-Pascal call for the Interface Customization feature:

```
VAR
infile :  PACKED ARRAY [1..12] OF CHAR ;
otfile :  PACKED ARRAY [1..11] OF CHAR ;
stat : INTEGER ;
intnum : INTEGER ;

infile := 'database.int' ;
otfile := 'newdata.int' ;

stat := nlxedt(infile[1], 12, intnum, otfile[1], 11) ;
```

## Link Stream

**C.4.2** To link a Pascal application program with the NaturalLink Sessioner, you must include both the NaturalLink Window Manager and the NaturalLink Sessioner software. Following is an example of a link control file for the MS-DOS Linkage Editor.

| | |
|---|---|
| LIMSPSH+ | Memory organization module **must** occur first |
| mymain+ | The Pascal application's main program |
| ...+ | Any application subroutines used |
| ...+ | Any NaturalLink called application routines used |
| ... | Modules for optional NaturalLink features |
| mymain /M | The .EXE file |
| mymain.MAP | The .MAP file |
| mylib+ | Any application libraries used |
| LIMSP+ | Library of High-Level Language Interface routines to NaturalLink |
| NL+ | NaturalLink Sessioner object library |
| WM+ | Window Manager object library |
| APPOTH+ | Dummy NaturalLink called application routines |
| Pascal | Pascal run-time library |

The APPOTH library contains dummy routines for all application routines called by NaturalLink. These include NLXEXP, NLXFUN, APPVAL, APPRCV, and so on. If you do not use all the features provided by these routines, the dummy modules in this library will resolve the references to them. If you have your own versions of these routines, they must be explicitly linked in following your main program. If you create libraries for any of your application routines, these libraries **must** be included before any NaturalLink libraries in the link stream.

If you choose to implement the Saved Sentences feature, you must include the module NLSAVQRY.OBJ in your link stream.

If you implement the Dynamic Lexical Items feature, include the modules NLDTLIB.OBJ and NLISLIB.OBJ in the link stream.

If you plan to include the Interface Customization feature, include the modules NLEDLIB.OBJ, NLSYLIB.OBJ, and NLISLIB.OBJ. If you do not wish to support Adding Synonyms but only the Editing Phrases function of the Interface Customization feature, do not include the NLSYLIB.OBJ module. If you do not wish to support the Interface Customization feature at all, do not include any of these three modules.

NLISLIB.OBJ is used for both the Dynamic Lexical Items feature and the Interface Customization feature. You must include this module if you use either of these features.

The order of this link stream is extremely important. Do not alter the order of the modules and libraries shown in the example when linking your application. All modules in this link stream must be included, regardless of the NaturalLink features implemented. Refer to Appendix E, Pascal Interface, in the *Window Manager Reference Manual* for more information on linking.

## Callable Routines — FORTRAN Interface

**C.5** The following paragraphs describe the specific calling sequence and linking information for MS-FORTRAN.

### Calling Syntax

**C.5.1** The routine declarations for the NL routines are located in the file called NLEXTRNS.FOR. You must include them in the application program prior to using calls to the routines.

An example of a call to the NaturalLink Sessioner from FORTRAN follows.

```
character xlatn(128)
character restxt(256)
integer*2 stat
integer*2 intnum

stat = nlload('database.int', 12, 'saves.svs', 9, intnum)

stat = nldriv(intnum, xlatn, 128, restxt, 256)

stat = nlxunl(intnum)
```

It is recommended that you place an array of size 1 in the `restext` parameter and a length of 0 (zero) if you do not want the actual text of the command sentence. This will free memory space that would be used to store this string.

The NLXEXP routine should be defined as follows:

```
  integer*2 function nlxexp(exptyp)
  integer*2 exptyp
```

Calls to the NLXEXP support routines appear as follows:

```
integer*2 stat
character xlatn(80)
character restext(80)

stat = nlxold(xlatn, 80)

stat = nlsetr(xlatn, 80, restxt, 80)
```

The NLXFUN routine should be defined as follows:

```
  integer*2 function nlxfun(keycod)
  integer*2 keycod
```

A library containing dummy routines for NLXEXP and NLXFUN has been provided. If you do *not* use these routines, the dummy routines resolve NaturalLink references to them.

Following is an example of the MS-FORTRAN call for the Dynamic Lexical Items feature:

```
character*12 infile
character*12 otfile
character*1  iffile
integer*2 stat
integer*2 intnum
character*7 window
character*15 item
character*6 tran

window = 'FIELDS:'
item = 'employee number'
tran = 'EMPNUM'
stat = nlxsnd(14,2,window, 7, item, 15, tran, 6)

item = 'address'
tran = 'EMPADD'
stat = nlxsnd(14,2,window, 7, item, 7, tran, 6)

item = 'name'
tran = 'EMPNAM'
stat = nlxsnd(14,2,window, 7, item, 4, tran, 6)

infile = 'database.int'
otfile = 'newdata.int'

stat = nlxcre(infile,12,intnum,iffile,0,otfile,11,0,0)
```

**NOTE:** The NLXSND call in FORTRAN differs from the call in C because an array of character pointers cannot be passed in FORTRAN. A separate NLXSND call must be made for each dynamic lexical item to be created.

Following is an example of the MS-FORTRAN call for the Interface Customization feature:

```
integer*2  stat
integer*2  intnum

stat = nlxedt('database.int', 12, intnum, 'newdata.int', 11)
```

**Link Stream**

**C.5.2** To link a FORTRAN application program with the NaturalLink Sessioner, you must include both the NaturalLink Window Manager and the NaturalLink Sessioner software. Following is an example of a link control file for the MS-DOS Linkage Editor.

| | |
|---|---|
| LIMSFSH + | Memory organization module **must** occur first |
| mymain + | The FORTRAN application's main program |
| ... + | Any application subroutines used |
| ... + | Any NaturalLink called application routines used |
| ... + | Modules for optional NaturalLink features |
| WMCINI | FORTRAN subroutine to init WMCOM |
| mymain /M | The .EXE file |
| mymain.MAP | The .MAP file |
| mylib + | Any application libraries used |
| LIMSF + | Library of High-Level Language Interface routines to NaturalLink |
| NL + | NaturalLink Sessioner object library |
| WM + | Window Manager object library |
| APPOTH + | Dummy NaturalLink called application routines |
| FORTRAN | FORTRAN run-time library |

The APPOTH library contains dummy routines for all application routines called by NaturalLink. These include NLXEXP, NLXFUN, APPVAL, APPRCV, and so on. If you do not use all the features provided by these routines, the dummy modules in this library will resolve the references to them. If you have your own versions of these routines, they must be explicitly linked in following your main program. If you create libraries for any of your application routines, these libraries **must** be included before any NaturalLink libraries in the link stream.

If you choose to implement the Saved Sentences feature, you must include the module NLSAVQRY.OBJ in your link stream.

If you implement the Dynamic Lexical Items feature, include the modules NLDTLIB.OBJ and NLISLIB.OBJ in the link stream.

If you plan to include the Interface Customization feature, include the modules NLEDLIB.OBJ, NLSYLIB.OBJ, and NLISLIB.OBJ. If you do not wish to support Adding Synonyms but only the Editing Phrases function of the Interface Customization feature, do not include the NLSYLIB.OBJ module. If you do not wish to support the Interface Customization feature at all, do not include any of these three modules.

NLISLIB.OBJ is used for both the Dynamic Lexical Items feature and the Interface Customization feature. You must include this module if you use either of these features.

The order of this link stream is extremely important. Do not alter the order of the modules and libraries shown in the example when linking your application. All modules in this link stream must be included, regardless of the NaturalLink features implemented. Refer to Appendix F, FORTRAN Interface, in the *Window Manager Reference Manual* for more information on linking.

## Callable Routines — BASIC Compiler Interface

**C.6** The following paragraphs describe the specific calling sequence and linking information for the MS-BASIC compiler.

### Calling Syntax

**C.6.1** The calling sequences for the NaturalLink routines are defined here.

**CAUTION: The BASIC compiler does not require external declarations of routines. You must be very careful that the calls to the NaturalLink routines are coded correctly, with the proper number of variables, and that each is in the correct position. BASIC does not check these factors, as other high-level languages do.**

An example of a call to the NaturalLink Sessioner from the MS-BASIC compiler follows.

```
DEFINT A-Z
XLATION$=SPACE$(128)      ' override DEFINT with explicit "$"
RESTEXT$=SPACE$(256)      ' appended to variable

   CALL NLLOAD("database.int", "saves.svs", INTNUM, STAT)

   CALL NLDRIV(INTNUM, XLATION$, RESTEXT$, STAT)

   CALL NLXUNL(INTNUM, STAT)
```

It is recommended that you place a length of 0 in the `RESTEXT$` parameter if you do not want the actual text of the command sentence. This will free memory space that would be used to store this string. To do this, use the following command:

```
RESTEXT $=" "
```

## Application Assembly Language Routines—NLXEXP and NLXFUN

**C.6.2** Details of how to write assembly language routines called by NaturalLink can be found in Appendix G, Compiled BASIC Interface, of the *Window Manager Reference Manual.*

Calls to the NLXEXP support routines appear as follows:

```
DEFINT A-Z
OLDEXP$=SPACE$(256)
XLATION$=SPACE$(256)
RESTEXT$=SPACE$(256)

   CALL  NLXOLD(OLDEXP$,STAT)

   CALL  NLSETR(XLATION$,RESTEXT$,STAT)
```

A library containing dummy routines for NLXEXP and NLXFUN has been provided. If you do *not* use these routines, the dummy routines resolve NaturalLink references to them.

Following are examples of the MS-BASIC compiler call for the Dynamic Lexical Items feature.

```
-- NLXSND call --

DEFINT A-Z
WIND$="FIELDS:"
ITEM$="employee number"
TRAN$="EMPNUM"

   CALL NLXSND(14, 2, WIND$, ITEM$, TRAN$, STAT)

ITEM$="address"
TRAN$="EMPADD"

   CALL NLXSND(14, 2, WIND$, ITEM$, TRAN$, STAT)

ITEM$="name"
TRAN$="EMPNAM"

   CALL NLXSND(14, 2, WIND$, ITEM$, TRAN$, STAT)


-- NLXCRE call --

Information sent; Leave Interface in Memory

DEFINT A-Z
NULL$=" "

   CALL NLXCRE(NULL$, INTNUM, NULL$, NULL$, 0, 0, STAT)

Load Dynamic Lexical Items, Save Interface, and Unload
Interface

DEFINT A-Z
INTFILE$="database.int"
INFOFIL$="database.dat"
OUTFILE$="newfile.int"

   CALL NLXCRE(INTFILE$, INTNUM, INFOFIL$, OUTFILE$, 1, 0,
   STAT)
```

**NOTE:** The NLXSND call in BASIC differs from the call in C because an array of character pointers cannot be passed in BASIC. A separate NLXSND call must be made for each dynamic lexical item to be created.

Following is an example of the MS-BASIC compiler call for the Interface Customization feature.

```
DEFINT A-Z
INTFILE$="database.int"
OUTFILE$="newdata.int"

   CALL NLXEDT(INTFILE$, INTNUM, OUTFILE$, STAT)
```

**Link Stream**

**C.6.3** To link a compiled BASIC application program with the NaturalLink Sessioner, you must include both the Window Manager and the Sessioner software. Following is an example of a link control file for the MS-DOS Linkage Editor.

| | |
|---|---|
| LIMSBSH+ | Memory organization module **must** occur first |
| mymain+ | The BASIC application's main program |
| ...+ | Any application subroutines used |
| ...+ | Any NaturalLink called application routines used |
| ... | Modules for optional NaturalLink features |
| mymain /M | The .EXE file |
| mymain.MAP | The .MAP file |
| mylib+ | Any application libraries used |
| LIMSB+ | Library of High-Level Language Interface routines to NaturalLink |
| NL+ | NaturalLink Sessioner object library |
| WM+ | Window Manager object library |
| APPOTH+ | Dummy NaturalLink called application routines |
| BASCOMG | Compiled BASIC run-time library (or BASRUNG) |

The APPOTH library contains dummy routines for all application routines called by NaturalLink. These include NLXEXP, NLXFUN, APPVAL, APPRCV, and so on. If you do not use all the features provided by these routines, the dummy modules in this library will resolve the references to them. If you have your own versions of these routines, they must be explicitly linked in following your main program. If you create libraries for any of your application routines, these libraries **must** be included before any NaturalLink libraries in the link stream.

If you choose to implement the Saved Sentences feature, you must include the module NLSAVQRY.OBJ in your link stream.

If you implement the Dynamic Lexical Items feature, include the modules NLDTLIB.OBJ and NLISLIB.OBJ in the link stream.

If you plan to include the Interface Customization feature, include the modules NLEDLIB.OBJ, NLSYLIB.OBJ, and NLISLIB.OBJ. If you do not wish to support Adding Synonyms but only the Editing Phrases function of the Interface Customization feature, do not include the NLSYLIB.OBJ module. If you do not wish to support the Interface Customization feature at all, do not include any of these three modules.

NLISLIB.OBJ is used for both the Dynamic Lexical Items feature and the Interface Customization feature. You must include this module if you use either of these features.

The order of this link stream is extremely important. Do not alter the order of the modules and libraries shown in the example when linking your application. All modules in this link stream must be included, regardless of the NaturalLink features implemented. Refer to Appendix G, Compiled BASIC Interface, in the *Window Manager Reference Manual* for more information on linking.

# NATURALLINK DEMONSTRATION PROGRAM D

## Introduction

**D.1** The NLLCDEMO program provided with your NaturalLink Toolkit software is a Lattice C program that demonstrates how to use the NaturalLink interface and the Window Manager. It will run with any interface file generated by IBUILD and can be used as an interface testing program.

## Toolkit and Window Manager Demonstration Program

**D.2** The screen associated with this program, NLLCDEMO.NS$, contains six windows:

0) Header window — Identifies the program

1) Available Topics window — Determines the interface file to load

2) Press window — Identifies the function keys available within the NaturalLink Sessioner, aside from the usual Window Manager keys

3) Translation window — Displays the translation generated

4) Sample Application window — Shows what an application might do upon receiving the translation from the NaturalLink Sessioner

5) File Information window — Requests filenames when the Specific Topic of Your Choice item is chosen from window 1

You can use this program to tailor your own demonstration by manipulating the Available Topics window and the Sample Applications window. You can modify the latter window, using the Screen Builder utility (SBUILD), to show what your specific application program will return upon receiving a translation. Be sure that at least one selectable item is visible in the window so that the Receive call will work correctly. If you want only to display data, change the window's type to text.

The Available Topics window has a special purpose in this program: it can be modified to load specific interfaces. As shipped, the window contains three items:

- 0, which is the label Available Topics
- 1, which has no text specified
- 2, which has the text < a specific topic of your choice >

## How to Modify the Available Topics Window

**D.2.1** Follow these steps if you have interface files that you want to demonstrate:

1. Run SBUILD and load the screen NLLCDEMO.NS$.

2. Add only two items per interface file to the item table in the Available Topics window. You can add them before or after item 2 < a specific topic of your choice > .

3. Make the first item you added invisible and unselectable with no item text.

4. Make the second item you added selectable with the item text describing the topic you wish to demonstrate (for example, "Jobshop").

5. Edit the item label of the first item you added to be the name of the saved sentences file for the interface you want to demonstrate (for example, JOBSHOP.SVQ).

6. Edit the item label of the second item you added to be the name of the interface file you want to demonstrate (for example, JOBSHOP.INT).

Thus, the items for the Available Topics window should resemble the following:

0) Available Topics

1) unselectable, no item text, no item label

2) unselectable, no item text, item label: JOBSHOP.SVQ

3) selectable, item text: Jobshop, item label: JOBSHOP.INT

4) selectable, item text: < a specific topic of your choice > , no item label

You can add as many interfaces as necessary to the list of available topics using this method.

When you bring up the demonstration program, the interfaces you want to demonstrate should be listed along with the < a specific topic of your choice > item. If you select the < a specific topic of your choice > item, a window will be displayed prompting you for the interface filename and the saved sentences filename for that interface. After you specify the interface and saved sentences filenames, the program will call the NaturalLink Sessioner and start the sentence-building process. If you select one of the interfaces you specified, the program will not ask for the interface and saved sentence filenames; the program will use the filenames already specified for the chosen interface.

### Demonstration Program Algorithm

**D.2.2** The algorithm of the program is as follows:

```
Make the necessary WM initialization call
Load the demonstration program screen
Until it's time to stop
   Add the demo program main screen (windows 0 through 2)
   Select the Available Topics window
   Receive from the Available Topics window
   Release the Available Topics window
   If the topic chosen didn't have interface and saved
         sentences filenames already associated with it
         (the < a specific topic of your choice> item)
      Add the File Information window
      Select the File Information window
      Receive from the File Information window
      Delete the File Information window
   Delete the main screen windows
   Call the NaturalLink interface loader
   Call the NaturalLink Sessioner
   If a translation was returned and the user didn't press
         the Quit key
      Add the Translation window (window 3)
      Select the Translation window
      Receive from the Translation window
      Delete the Translation window
      Add the Sample Application window (window 4)
      Select the Sample Application window
      Receive from the Sample Application window
      Delete the Sample Application window

— End of the "Until it's time to stop" loop —
Unload the screen
Call the NaturalLink interface unloader
Make the required WM reset call
Clear the screen
Exit
```

The NLLCDEMO.C source file also contains a routine called NLXEXP, which is the routine the Sessioner calls if a user-defined expert is encountered. A routine called NLXEXP must be present in any application using the NaturalLink Sessioner, whether or not any user-defined experts were used, to resolve external references. A library containing a dummy NLXEXP routine has been provided and should be linked in to resolve the external references if an application NLXEXP is not present. The NLXEXP routine algorithm is:

```
Load the expert handling screen (NLLCEXP.NS$)
Get the window label for the Expert window
Put the expert code parameter into the window label
Add the Expert window
Select the Expert window
Receive from the Expert window
Delete the Expert window
Restore the original window label to the Expert window
Unload the expert handling screen
If a special-purpose function key was pressed
   ENTER the appropriate status code
Call NLSETR to pass the expert value received to the NaturalLink
      Sessioner
Return to the NaturalLink Sessioner
```

## Linking NLLCDEMO

**D.2.3** NLLCDEMO is provided in Lattice C source and EXE form (NLLCSDEM.EXE). If you want to compile the NLLCDEMO program yourself, use the Lattice C compiler. The compiler will require the two files STDIO.H and WMFIELD.H during compilation. Any of the four Lattice C models—S (small), P (program), D (data), or L (large)—can be used. Replace the question mark (?) in the link stream and EXE file pathnames with the letter corresponding to the model used in compiling the NLLCDEMO.C source file—for example, S, if the small model was used.

To link the NLLCDEMO object with the Window Manager and NaturalLink Sessioner object, use the NLLC?DEM.LNK link control file with the MS-DOS linker to produce NLLC?DEM.EXE. The linker requires that the following files be present during the linking process:

- Toolkit object files
  - NLSAVQRY.OBJ
  - NLSYLIB.OBJ
  - NLEDLIB.OBJ
  - NLISLIB.OBJ
  - NLDTLIB.OBJ
- Window Manager object file
  - LILC?SH.OBJ
- Window Manager and Toolkit Libraries
  - NL.LIB
  - WM.LIB
  - LILC?.LIB
  - APP???.LIB (LCS, LCD, OTH—for P and L models)
- NLLCDEMO.OBJ file
- Lattice C C?.OBJ file
- Lattice C run-time library LC?.LIB,

The following commands invoke the linker.

- LINK @NLLCSDEM.LNK (for small model)
- LINK @NLLCPDEM.LNK (for program model)
- LINK @NLLCDDEM.LNK (for data model)
- LINK @NLLCLDEM.LNK (for large model)

Once you have accessed the NLLC?DEM.EXE file, either by linking up the NLLCDEMO program yourself or by copying the file directly from the Toolkit object disk, the following commands invoke the program.

- NLLCSDEM (for small model)
- NLLCPDEM (for program model)
- NLLCDDEM (for data model)
- NLLCLDEM (for large model)

Be sure that the following files are on the default disk when you run the NLLC?DEM program.

- NLLCDEMO.NS$
- NLLCEXP.NS$
- LIERRMSG.NM$

## Defining the Quit Key

**D.2.4** The Quit key is mentioned in several of the windows for the NLLCDEMO program. In the Window Manager Function Key Change utility, this key is called the Quit or Abort key. If you link the demo program with the default WMKEYDEF.OBJ module included in the Window Manager library WM.LIB (the NLLCSDEM.EXE file provided is linked with the default), the Quit key will be defined as the **ESC** key. If you use the Function Key Change utility to modify the key used for Quit and link that object file with the demonstration program, the new key that you define will be the Quit key for the NaturalLink Sessioner portion of the demonstration only; the rest of the demo will still use the **ESC** key, since the key is defined in the NLLCDEMO program code. To change the Quit key for the rest of the demonstration program, change the following define statement to show the key code for your new Quit key:

```
#define QUIT_KEY 0x01B.
```

# GLOSSARY

## a

| | |
|---|---|
| **abbreviatory symbol** | One of the three types of symbols used to collapse a number of rules, each sharing the same mother and left corner, into a single rule. The abbreviatory symbols used by NaturalLink grammars are parentheses, braces, and square brackets. In the expansion of a grammar rule:<br>■ Parentheses mean that the application of the rule elements within is optional.<br>■ Braces mean that one of the elements within must be applied.<br>■ Brackets mean that the rule elements within are to be applied as a group. |

## c

| | |
|---|---|
| **Command window** | A window that contains one or more of the nine NaturalLink command options (for example, Execute, Customize, Quit, and so on). |
| **committing a window** | Pressing the Proceed key in a multiple-selection window or **ENTER** from the last line of a multiple-selection edit window. |
| **context-free grammar** | In a context-free grammar every rule has the form of: A $\longrightarrow$ X<br>where:<br>A is a single nonterminal element and X is a non-null string of elements. A $\longrightarrow$ X may be read as 'A' consists of 'X'. |

## d

| | |
|---|---|
| **daughter** | In each grammar rule in the NaturalLink grammar format, all rule elements that appear after the mother are daughters: @< mother > < daughter1 > ....< daughterN > |
| **DTS (dynamic terminal symbol)** | A terminal element that was removed or is to be removed from the lexicon by means of the Dynamic Lexical Items feature. |
| **Dynamic Lexical Items feature** | The NaturalLink feature that allows the user to make limited modifications to the lexicon and the screen of an interface through an application program. |

## e

| | |
|---|---|
| **expert** | A routine that handles a specific task—such as input and validation of literal values. NaturalLink software enables the use of four standard experts: alphanumeric type-in, numeric type-in, date, and list. A fifth type of expert—the user-defined expert—can be defined within an application and associated with the NaturalLink menu screen (NLmenu screen). |

## f

| | |
|---|---|
| **fatal error** | A nonrecoverable error. You should stop processing and resolve the problem. |

## g

| | |
|---|---|
| **grammar** | The set of rules that govern how words can be put together to form sentences. The grammar used with NaturalLink software must produce all, and only, the sentences required by a particular application. |

## i

| | |
|---|---|
| **IBUILD** | The NaturalLink Interface Builder utility. This utility aids in creating and testing the NaturalLink grammar, lexicon, and interface. |
| **interface file** | A binary file that contains the grammar, lexicon, and screen description for a NaturalLink application. |
| **item** | An entry within a NaturalLink window. |

## k

| | |
|---|---|
| **KBUILD** | The Window Manager Function Key Change utility. This utility lets you change the default function keys used by Window Manager. |

## l

| | |
|---|---|
| **left corner** | The first rule element on the right side of a grammar rule (also called daughter 1). The NaturalLink grammar format does not permit the left corner to be enclosed by abbreviatory elements. |
| **lexical item** | A terminal element in the grammar that will be further defined in the lexicon. |
| **lexicon** | The dictionary of the NaturalLink system. It relates each terminal element in the grammar to a particular natural language word or phrase, to a location on the screen, and to a translation in the target language. |

## m

| | |
|---|---|
| **MBUILD** | The NaturalLink Message Builder utility. This utility aids in constructing messages for use in an application program. |
| **mother** | In the NaturalLink grammar format, the first rule element in a grammar rule is the mother. |
| **NLmenu** | The interface screen that presents the words and phrases the user can select to build a command sentence. |

## n

| | |
|---|---|
| **nonterminal element** | A grammar element that appears on the left side of rules and usually appears on the right side also. |

## p

| | |
|---|---|
| **Parser** | The component of the NaturalLink software that resolves a user's choice of a word or phrase from a menu into its correct grammatical element and determines, based on the grammar, what the user's next legal choice(s) will be. |
| **parse tree** | A graphic representation of the structure of a sentence. Each rule used in the parse is depicted as a subtree that has the mother as the root node and the daughters as the child nodes. The root of the parse tree is the start symbol of the grammar, and the leaf nodes (those nodes with no children) are the terminal elements in the sentence. |
| **parse window** | A window that contains the words and/or phrases the user selects to build NaturalLink command sentences. |
| **PBUILD** | The NaturalLink Phrase Editor utility. This utility lets you change the internal phrases of NaturalLink. |
| **predicate** | The first integer following the opening parenthesis in the semantic path list. The predicate has a character string associated with it called the predicate's string. The predicate's string contains "slots" in which arguments in the semantic substitution list are placed. |

## r

| | |
|---|---|
| **Results window** | A window that displays the command sentence as the user builds it. |

## s

| | |
|---|---|
| **SBUILD** | The NaturalLink Screen Builder utility. This utility aids in constructing screens for use with Window Manager. |
| **screen** | A collection of NaturalLink windows. |
| **screen description** | The binary file that contains all the information necessary for the NaturalLink Window Manager to draw and manipulate a collection of windows. |
| **select** | A function of the NaturalLink menu screen. The user places the cursor on the item desired and presses the **ENTER** key. |

| | |
|---|---|
| **semantic substitution list** | The sequence of integers that represents the ordering of the target language translation. It is the second sequence of integers appearing in a grammar rule's translation list. |
| **semantics** | The study of meanings. Semantic rules determine whether sentences are meaningful. |
| **Sessioner** | The main driver of the NaturalLink user software. It handles all interaction among the Window Manager, the Parser, the Translator, and the target application. |
| **slots** | Locations in the predicate's string where arguments are placed. Slots are indicated by an at sign (@) followed by an integer in the predicate's string. |
| **start symbol** | The first element of the first rule in a NaturalLink grammar. In the example NaturalLink grammars, the symbol 'S' is used as the start symbol. |
| **syntactic path list** | The sequence of integers that identifies which expansion of a grammar rule is to be translated. It is the first sequence of integers appearing in a grammar rule's translation list. |
| **syntax** | The set of rules that governs the legal order of words within a language. |

## t

| | |
|---|---|
| **target language** | The language used by the application behind the NaturalLink software, such as the command control language in a computer-controlled industrial system or the data manipulation language of a database. |
| **target string** | The target translation of a terminal element in a NaturalLink grammar. |
| **terminal element** | A grammar element that appears only on the right side of the rule. |
| **translation lists** | The sets of numbers in parentheses that appear at the end of each grammar rule. There is a separate translation list for each expansion of a grammar rule. Each translation list consists of a semicolon and two sets of parentheses. The numbers in the first set of parentheses are the syntactic path list; they describe how to form one expansion of the grammar rule. The numbers in the second set of parentheses are the semantic substitution list; they tell the translator how to combine the individual translation strings to create the correct target-language command. |
| **Translator** | The component of the NaturalLink software that, by consulting the lexicon, resolves a completed parse tree into the target language command. |

## u

| | |
|---|---|
| **UDW (user-defined window)** | A special window specified in Screen Builder. The application has full control of any display, receive, and delete functions performed on a UDW. |

## w

| | |
|---|---|
| **warning code** | Indicates incorrect use of some NaturalLink feature. The code can be ignored, but you should check the problem during the development process. |
| **well-formed grammar** | A term for a grammar that is free of syntactic and semantic errors. |
| **window** | A rectangular area on the video display terminal. Windows in a NaturalLink screen can be bordered or unbordered. |

# INDEX

## USER RESPONSE SHEET

Our manuals are written for you. Please help us improve them by completing and mailing this form. List any discrepancy or other problem that you found in this manual by page, paragraph, figure, or table number in the space provided, and add any suggestions you have. Thank you.

**Manual Title:** ______________________________

**Manual Date:** ______________ **Today's Date:** ______________

**Your Name:** ______________ **Telephone:** ______________

**Company:** ______________ **Department:** ______________

**Company Address:** ______________________________

**Product Purchased From:** ______________________________

**Company** **City** **State**

**Location in Manual** **Comment/Suggestion**

**The manual's good points:**

**What needs to be improved:**

**No postage necessary if mailed in U.S.A.**
**Fold on two lines (located on reverse side), tape and mail**

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DATA SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD

Cover Part No. 2310002-0001

TEXAS INSTRUMENTS