IBM Dictionary and Linguistic Tools

SC30-4040-00

User's Guide for the Testing Tool

Version 2.7

Note

Before using this information and the product it supports, be sure to read the general information under Appendix A, “Notices” on page 91.

First Edition (May 2000)

This edition applies to Version 2.7 of the IBM Dictionary and Linguistic Tools.

IBM welcomes your comments on this publication. A form for readers’ comments is included at the back of this publication. You can also address comments to:

Department CGF
Design & Information Development
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC 27709

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About this manual

This manual describes the testing tool used in conjunction with the IBM Dictionary and Linguistic Tools.

Note: The short name for the IBM Dictionary and Linguistic Tools is Linguistic Tools.

Who should read this manual

This manual is intended for software developers who work on products that use or plan to use Linguistic Tools.

Conventions used in this manual

The following conventions are used in this manual:

Convention	Usage	Example
Italics	Variables shown in the text, not in the code samples.	*dictionary_filename*
Bold	Commands (functions) shown in the text, not in the code samples. **Note:** Programmed commands are not case sensitive.	**Activate**
Monospaced text	Code samples	`/DLX_WIN`
All capitals	Keywords	LX_UCHAR
Quotation marks	Words out of context, as well as quotations	Neither “a” nor “an” ...

Related Publications

The companion manual that describes the application programming interface for the Linguistic Tools is:

IBM Dictionary and Linguistic Tools Application Programming Interface Description, SC30-4039.

Chapter 1. Overview

General Description

The IBM Dictionary and Linguistic Tools Testing Tool (from hereon called Linguistic Tester or just tester), is an application that allows a user to access and test the functions available in the IBM Dictionary and Linguistic Tools (called Linguistic Tools). The Linguistic Tester functions may be executed either interactively or from commands written in a batch input file. Output may be displayed either on the screen or written to a file.

Unless an input filename is provided, the Linguistic Tester will receive commands from the keyboard. It will evaluate each command until either the end of the input file is reached, or the user issues the **Quit** command from the keyboard.

Tables 1, through 3 represent the functions/commands supported by the Linguistic Tools or Linguistic Tester. The tables are categorized by Linguistic Tools functions, Linguistic Tools utilities, and commands used specifically by the Linguistic Tester, respectively. The right-hand column of each table represents the command word used to invoke the function. Examples and further descriptions of each command can be found in Chapter 4, “Command Description” on page 15.

Table 1 (Page 1 of 2). Linguistic Tools Functions

Description	Command
Activate Dictionary	**Activate**
Add Morphology to Addenda	**AddMorph**
Add Word to Addenda	**AddWord**
Article Checking	**ArtCheck**
Compound Word Component Isolation	**Compound**
Create Addenda	**Create**
Deactivate Dictionary	**Deactivate**
Dehyphenate	**Dehyphenate**
Extract Keywords	**EKW**
Extract Query Terms	**EQT**
Fuzzy Spell Aid	**FuzzySpellAid**
Generate Inflected Forms	**GIF**
Grade Level Analysis	**Grades**
Hyphenate	**Hyphenate**
Inflect Word from Model	**InflectWord**
Initialize	**Initialize**
Lexical Analysis	**Complex**
List Addenda Words	**ListWords**
Look Up Word	**Lookup**
Morphological Identification	**MID**

Table 1 (Page 2 of 2). Linguistic Tools Functions

Description	Command
Remove Word from Addenda	**Remove**
Save Addenda	**SaveAdd**
Simple Tokenization	**Simple**
Single-Word Data Element Creation	**Element**
Spell Aid	**SpellAid**
Spell Verify	**Verify**
Synonym and Definition Aid	**Synonym**
Terminate	**Terminate**
Text Segment Identification	**TextSeg**

Table 2. Linguistic Tools Utilities

Description	Command
Check if Character Is A Delimiter	**IsDelim**
Find/List Available Dictionaries	**Avl_Dicts**
Query Default Dictionary Search Path	**QryPath**
Query Word Delimiter Table	**QryDelim**
Register Code Page	**RegCP**
Set Default Dictionary Extensions	**SetExts**
Set Default Dictionary Search Path	**SetPath**
Set/Reset Mixed Mode State	**MixedMode**
Set Word Delimiter Table	**SetDelim**

Table 3. Linguistic Tester-Specific Commands

Description	Command
Call a Tester Command File	**Call**
Continue Reply	**Continue**
Copy Reply Area Buffer	**CopyBuff**
Delete File(s) from Disk	**DelFile**
Display Current Dictionary List	**Current**
Execute Current Control Block Configuration	**Execute**
Execute System Command	**System**
Exit Tester	**Exit**, **Quit**
Generate Data Element List from Input File	**InFile**
List Available Tester Commands	**Help**
Modify/Display Control Block Fields	**CB**
Redirect Tester Output to File	**OutFile**
Specify Level of Tester Output Detail	**OutLevel**
Reset Control Block	**Reset**

Executing the Linguistic Tester

The following table shows the name of the executable file that invokes the Linguistic Tester:

Table 4. How to Invoke the Linguistic Tester

Platform	Executable Name
32-bit Microsoft Windows (Windows NT®, Windows 95®, Windows 98®)	tstnl27w
OS/2 Warp®	tstnl272
AIX® (IBM UNIX®)	tstnl27x
HP-UX (Hewlett-Packard UNIX)	tstnl27h
Sun® Solaris (Sun Microsystems UNIX)	tstnl27s
OS/390® OpenEdition (OS/390 UNIX System Services)	tstnl273
OS/400®	Library: QDICT Object: TSTNL274 Type: *PGM
Mac OS	tstnl27a

Under Mac OS, the Linguistic Tester may only be invoked by double-clicking on the "Linguistic Tools Interactive Tester" icon in the Finder window. Also, only interactive mode is supported. However, test scripts may still be run by use of the **Call** command.

Specifying an Input or an Output File

Specifying an optional filename parameter when invoking the Linguistic Tester, indicates to use that file for batch command input. If an optional filename is not provided, the Linguistic Tester, by default, will read input from the keyboard and write output to the screen. The **OutFile** command is a useful way of directing output to a file.

For all platforms, except the Macintosh, that support environment variables, a non-qualified command input file will be searched for in the current directory followed by every directory specified by the TESTPATH environment variable.

Chapter 2. Input to the Linguistic Tester

The Linguistic Tester uses two major forms of input: command input and text input. Commands may be input either interactively or through a batch command file. Text input may be provided interactively, through a tagged input file, or through a plain text input file. Each of these input conventions is described in the following section.

Command Input

Commands may be input either interactively or through a batch command file. Interactive input provides the capability for quick and simple testing. Batch command files are designed to allow large-scale, repeatable testing. Each of the commands supported by the Linguistic Tester is described in “Chapter 4, Command Description”, starting on page 15.

Interactive Input

The user may type commands directly to the Linguistic Tester if an input filename is not provided upon invocation. Commands will be interpreted by the Linguistic Tester until the user issues the **Quit** (or **Exit**) command.

Batch Input

Batch input files contain a series of command statements for controlling the Linguistic Tester. These command statements consist of keywords and arguments which tell the Linguistic Tester which function to invoke and how it should display the results of the function.

To use the command statements, you should create a flat file and insert one statement per line. You may also opt to insert blank lines and comment lines in this batch input file. The Linguistic Tester attempts to evaluate blank lines or lines that begin with the comment indicator, an asterisk (*). Commands from the batch input file will be executed until the end of file has been reached or a **Quit** (or **Exit**) command is encountered, whichever comes first.

Text Input

The Linguistic Tester will accept text input either interactively or from a text input (TIN) file.

Interactive Input

When text is entered immediately following a command that supports interactive text input, that text is placed into a single data element. This single data element replaces any data-element list that may already exist.

Input from a Text Input File

The **InFile** command sets a default text input file.

Text input files may be either tagged or plain text. Both forms of text input create a data-element list for use by the Linguistic Tools. While the tagged input format provides the ability to specify the contents of every data element, the plain text input format is simpler.

Tagged Input Format — Option /b of the InFile Command

The tagged input format provides a complete interface to create data elements for use by the Linguistic Tools. It should contain text that is interspersed with document formatting tags and may include optional comment lines. (Formatting tags are tags used with beginning and ending delimiters and define the look of the formatted document, such as, :p. to create a new paragraph.) The Linguistic Tester will divide the file into a list of consecutive data elements and then pass the pointer for the beginning of the data-element list to the Linguistic Tools, through the *lx_element_p* field of the Linguistic Tools control block.

Text in the input file may be separated into data elements by using the **:Element** tag. This tag has eleven optional parameters:

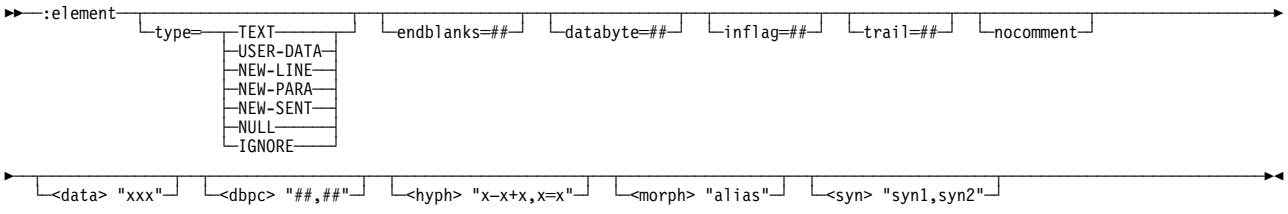
type=
endblanks=
databyte=
inflag=
trail=
nocomment
<**data**>
<**dbpc**>
<**hyph**>
<**morph**>
<**syn**>

The last five are used only for the **AddWord** command of of the Linguistic Tester or **AddWord to Addenda** function of the Linguistic Tools.

The diagram below shows the syntax of the **:Element** tag followed by descriptions of the optional parameters and examples for some of the parameters.

Points to Remember:

1. For those parameters that require an equals sign and a value, there should be no spaces on either side of the equals sign.
2. Numeric digits are represented by the pound character (#).
3. The angle brackets (< and >) and quotes (") are required input for those parameters showing them.
4. Although it is possible to put text into an input file that is written in a different code page, the **:Element** tag and its parameters ***must*** be in the native code page (850 or 500) of the platform on which the Linguistic Tester is being executed.



type= Defaults to the value of TEXT and tells the Linguistic Tester which type of data element to create. Other valid values for the type parameter are:

USER-DATA
NEW-LINE
NEW-PARA
NEW-SENT
NULL
IGNORE.

The Linguistic Tester places a value in the *lx_type* field of the data element, accordingly.

endblanks= Defaults to a value of 0 and is used to add blanks to the end of each line of this TEXT or USER-DATA element in the input file. This is useful when an editing program removes trailing blanks from the lines in an input file.

databyte= Defaults to a value of 0 and determines the value that is placed in the *lx_databyte* field of the data element.

inflag= Defaults to a value of 1 and determines the value that is placed in the *lx_input_f* field of the data element.

trail= Allows the user to indicate the character that follows the word (the trailing character) and specifies the value that will be placed in the *lx_trail* field of the data element. This parameter defaults to the value of a space.

nocomment Tells the Linguistic Tester to consider any line beginning with an asterisk (*) as actual input data instead of just a comment. This setting remains in effect until the next **:Element** tag is reached.

<data> Tells the Linguistic Tester to set up a word information structure for the data element, referenced by the *lx_info_p* field, and to take the value following the **<data>** parameter as user data. The user data must be enclosed in quotes and there must be a space between the **<data>** parameter and the leading quote.

<dbpc> Tells the Linguistic Tester to set up a word information structure for the data element, referenced by the *lx_info_p* field, and to take the value following the **<dbpc>** parameter as a list of Double-Byte Parse Codes (DBPCs). Individual DBPCs must be separated by commas, the complete DBPC data must be enclosed in quotes, and there must be a space between the **<dbpc>** parameter and the leading quote.

<hyph> Tells the Linguistic Tester to set up a word information structure for the data element, referenced by the *lx_info_p* field, and to take the value following the **<hyph>** parameter as hyphenation data. The hyphenation data must be a single word enclosed in quotes with marked hyphenation points and there must be a space between the **<hyph>** parameter and the leading quote.

<morph> Tells the Linguistic Tester to set up a word information structure for the data element, referenced by the *lx_info_p* field, and to take the value following the **<morph>** parameter as a morphology alias. The morphology alias must be a single word enclosed in quotes and

there must be a space between the **<morph>** parameter and the leading quote.

<syn> Tells the Linguistic Tester to set up a word information structure for the data element, referenced by the *lx_info_p* field, and to take the value following the **<syn>** parameter as a list of synonyms. Individual synonyms must be separated by commas, the complete synonym data must be enclosed in quotes, and there must be a space between the **<syn>** parameter and the leading quote.

When a TEXT or USER-DATA element is being created, all of the data on the lines following the tag is placed into a consecutive area in memory until the next **:Element** tag is encountered. However, any blank lines and those which begin with the comment delimiter (*) are ignored. The address of the area created to hold the data is placed in the *lx_data_p* field of the data element. The storage for this area is freed when a new **InFile** command is issued, a **Quit** (or **Exit**) command is issued, or the **Reset** command is issued.

If two **:Element** tags are placed on consecutive lines, special exceptions occur. If the first tag had an endblanks parameter, then an element is created that contains only blanks. Otherwise, an element is created whose *lx_data_p* pointer is equal to NULL.

```
* The following data element defaults to text with no trailing blanks
:element
Once upon a time, there were three

* This data element overrides the default type and creates a
* new-line data element.
:element type=new-line.

* The following creates a single text element and places two trailing
* blanks after "baby bear." and "forest."
:element endblanks=2.
bears: a mama bear, a papa bear, and a baby bear.
They lived in a house deep in the forest.

* This creates a text element equal to three spaces
:element endblanks=3

* This creates a text element equal to "5 * 3 = 15".
:element nocomment
5
* 3 = 15

* The following data element gives additional data to associate with
* the word "defenestrate"
:element <morph> "penetrate" <syn> "throw out the window"
defenestrate
```

Figure 1. Example of Tagged Input Format

Plain Text Input Format — Options /s and /f of the InFile Command

The plain text input format is designed to allow for easy, generalized text input to the Linguistic Tools. As in the tagged input format, blank lines and lines beginning with the asterisk (*) comment indicator will be ignored. All other lines are placed in separate data elements of type TEXT and whose *lx_trail* field equals a space.

The **/f** option creates a NEW-LINE data element following each input line of text and treats all lines in the file as data (an * does not indicate a comment). The **/s** option does **not** create a NEW-LINE data element following each input line of text and skips lines beginning with the asterisk (*) comment indicator.

```
* Each of the following words is placed in a separate text element
One
day
Goldilocks

* Plain text is not limited to single words
went into the forest to play.
```

Figure 2. Example of Plain Text Input Format

Chapter 3. Output from the Linguistic Tester

Each time the Linguistic Tools is invoked by the Linguistic Tester, an output header and other results of the processing are written to a file identified by the **OutFile** command statement. If an output filename has not been provided, output will be displayed on the screen. The contents of the header and the format of the results are outlined in the following sections.

Header

The header contains information that reflects the condition of the Linguistic Tester at the time the command was executed. The following fields are included in the header:

- Batch Input

  This field contains the name of the Batch Input File used in the current session.

- Text Input

  This field contains the name of the Text Input File, specified by the most recent **InFile** command, whose data was processed.

- Function

  This field contains the name of the function whose results are listed below the header.

- Return Code

  This field reflects, in words, the *lx_rc* field of the control block returned by the Linguistic Tools.

- Continue Reply Flag

  This field reflects, as a Y (for Yes or true) or an N (for No or false), the *lx_cont_reply_f* field of the control block returned by the Linguistic Tools.

Output Data

Output data from linguistic functions may be returned in one of three areas:

1. Data-element list
2. Token list
3. Reply Area

Each of these areas requires a separate output format. When output is returned in the reply area, function-specific routines are used to output the data.

Data-Element Output Format

The Data-Element Output Format is a record format that is common to most Dictionary Access Processor (DAP) functions. For each data element that was input, the Linguistic Tester will output the following information:

1. Return Code (RC)
2. Input Flag
3. Data Byte
4. Trailing Delimiter

5. Length (of the data)
6. Data

```
       Input    Data    Trailing
RC     Flag     Byte    Delimiter  Length    Data
----   -----    ----    ---------  -------   -----
 0     0x00     0xAF      0x20       4       Once
 0     0x00     0xAF      0x20       4       upon
 0     0x00     0xAF      0x20       1       a
 0     0x00     0xAF      0x20       4       time
```

Figure 3. Example of Data-Element Output Format

Token List Output Format

The Token List Output Format is required for the Text Analysis functions. It is designed to display all information stored in an output token list. This information includes

- Token Type
- Number of trailing blanks (called Right Blanks)
- String Flag (as a bit mask)
- Content Bits (as a bit mask)
- Type of leading character in the string (called Begin Type)
- String Length
- String Value
- A property name and property value for every property node or component associated with the token.

If a token is a joined token, both the joined token information and each component's information will be output. The joined characteristics of the token will be output first, with the character J in the first column. Next, each component's information will be output with the character C in the first column.

```
  Token  Right   String   Content  Begin  String  String
  Type   Blanks  Flag     Bits     Type   Length  Value
  -----  ------  ------   -------  -----  ------  ------
  0x03     0     0x00      0x00    0x00     0
  PROPERTY:  FIRSTELEM POINTER VALUE TO "Once u"...
  PROPERTY:  FRAGMENT

J 0x01     0     0x00      0x03    0x02     9      Once upon
C 0x01     1     0x00      0x03    0x02     4      Once
C 0x01     1     0x00      0x01    0x01     4      upon

  0x01     1     0x00      0x01    0x01     1      a

  0x01     1     0x00      0x01    0x01     4      time
```

Figure 4. Example of Token List Output

Token Property Output

The output information of a property in a token list will depend on the name of the property. If an unrecognized property is encountered, its numeric value will be output in integer format and the pointer value of the property will be output in hexadecimal format. The following is a list of recognized property symbol names:

- ABBREV
- CINF
- CKWD
- CLEM
- CSEN
- CSYN
- FIRSTELEM
- JPOS
- SUBTERM

ABBREV: The ABBREV property is placed on tokens by the **Text Segment Identification** function of the Linguistic Tools. Its value of either LX_NOTINDAP or a number is placed as part of the property-node instead of a pointer value for the property value. The Linguistic Tester will output the integer representation of the value.

CINF CKWD CLEM CSYN: The CINF, CKWD, CLEM, and CSYN properties are placed on a text token by the **EKW** and **EQT** commands. Every one of their values is a pointer to a structure which contains the following fields:

- Type
- Part-of-speech
- Length (of the normalized keyword)
- String (of the normalized keyword)
- Number (indicating the keyword order or compound component order)

CSEN: The CSEN property is placed on a NEW-SENT token by the **EKW** and **EQT** commands. Its value is the sentence number.

FIRSTELEM: The FIRSTELEM property is inserted by the **Text Segment Identification** function of the Linguistic Tools. Its value is a pointer to the data element which contains the first character of the text segment. The Linguistic Tester will output this information by computing the sequence number of that data element in the data-element list and displaying the first 6 bytes of that element (if the element is a TEXT or USER-DATA element).

JPOS: The JPOS property is placed on a TEXT token by the **Lexical Analysis** function of the Linguistic Tools for Japanese. Its value is an integer corresponding to the Japanese part of speech code. A token may have more than one JPOS property.

SUBTERM: The SUBTERM property is placed on a TEXT token by the **Lexical Analysis** function of the Linguistic Tools for Chinese when more than one parse is possible for a given portion of the input text. The value of this property indicates both the offset where the word begins and the length of the word. The offset is given with reference to the current token; an offset of 0 indicates the first byte of that token. The length is the number of bytes beginning from the specified offset. A token may have more than one SUBTERM property.

Other Properties: Other properties such as INITCAP, ALLCAP, ROMNUM, CPXHEAD, NOLOOKUP, and so on may be added to TEXT tokens by the **Lexical Analysis** function of the Linguistic Tools. All recognized properties are defined in one of the include files.

Reply Area Output Format

The Reply Area Output Format is specific to the function that produces it. For a detailed description of reply area output formats, refer to the individual command descriptions.

Chapter 4. Command Description

Command Summary

Activate	Opens a dictionary and adds it to the current dictionary list
AddMorph	Adds a word with its morphology to an addenda dictionary
AddWord	Adds a word and its associated data to an addenda dictionary
ArtCheck	Checks whether articles in the input text are correct
Avl_Dicts	Finds the dictionaries available on the system
Call	Executes Linguistic Tester commands contained in the specified file and returns
CB	Sets fields in the Linguistic Tools control block
Complex	Performs lexical analysis on the input text
Compound	Isolates components of a compound word
Continue	Continues reply of the previous request
CopyBuff	Copies the Reply Area to a buffer
Create	Creates an addenda dictionary and adds it to the current dictionary list
Current	Displays the current dictionary list
Deactivate	Closes a dictionary and removes it from the current dictionary list
Dehyphenate	Dehyphenates a word
DelFile	Deletes file(s) from disk
EKW	Extracts keywords from the input text
Element	Creates single-word data elements from the input text
EQT	Extracts query terms from the input text
Execute	Calls the Linguistic Tools with the current control block configuration
Exit	Ends Linguistic Tester session (same as **Quit**)
FuzzySpellAid	Obtains spell aid candidates for an input word based on a fuzzy word search argument
GIF	Generates morphologically inflected forms of the input word
Grades	Returns grade level information for the input words
Help	Displays a list of recognized Linguistic Tester commands
Hyphenate	Obtains hyphenation points for a word
InFile	Sets the default text input file
InflectWord	Generate morphologically-inflected forms of the input word based on the inflection scheme for the model word
Initialize	Initializes a Linguistic Tools conversation

IsDelim	Determines whether a character is a word delimiter
ListWords	Lists words in addenda dictionary
Lookup	Looks up a word and its associated data in a dictionary
MID	Return morphological information for the input word
MixedMode	Sets/Resets all-uppercase-matches-mixedcase mode flag
OutFile	Sets the default output file
OutLevel	Specifies the level of detail output by functions
QryDelim	Displays the current delimiter table used by the Linguistic Tools
QryPath	Displays the current dictionary search path
Quit	Ends Linguistic Tester session (same as **Exit**)
RegCP	Registers a codepage translation table with the Linguistic Tools
Remove	Removes a word from an addenda dictionary
Reset	Resets the control block to default values
SaveAdd	Saves changes to an addenda dictionary
SetDelim	Changes the current delimiter table used by the Linguistic Tools
SetExts	Changes the default dictionary filename extensions
SetPath	Changes the current dictionary search path
Simple	Performs simple tokenization on the input text
SpellAid	Obtains spell aid candidates for a word
Synonym	Obtains synonyms for a word
System	Executes a system command
Terminate	Terminates the Linguistic Tools conversation
TextSeg	Performs text segment identification on the input text
Verify	Verifies the spelling of words

Command Syntax Description

The following conventions are used for the syntax of each command shown in this section.

- All required items are underlined.
- Input variables that must be replaced by actual data are *italicized*.
- Choices are separated by a comma — when choices are given, only one should be selected and the comma should not be entered as part of the command.
- Numeric digits are represented by the pound character (#).
- Commands are shown in bold with the minimum number of required characters shown in upper case.
- The command prompt for the Linguistic Tester is a single period (.) and is not shown in the examples for each command.

The Linguistic Tester will convert all commands and command options to uppercase before interpreting the commands. Any text included on the command line for use by the Linguistic Tools will remain unchanged.

Activate

SYNTAX

ACTivate *dictionary_filename*

DESCRIPTION

The **Activate** command allows the user to specify a dictionary for use.

The Linguistic Tester maintains the current dictionary token list in the Linguistic Tools control block. A dictionary is added to the dictionary token list only if it was successfully activated by the Linguistic Tools (return code = LX_RC_OK). If the dictionary is a stop-word dictionary, its token is added to the end of *lx_stopw_tkns* and *lx_stopw_tkns_ct* is incremented. If the dictionary is an abbreviation dictionary, its token is added to the end of *lx_abbr_tkns* and *lx_abbr_tkns_ct* is incremented. Otherwise, the dictionary token is added to *lx_dict_tkns* and *lx_dict_tkns_ct* is incremented.

Input Format

The only input to the **Activate** command is the name of the dictionary to be activated. However, only one dictionary may be specified. The fully-qualified dictionary filename may be entered. When the **Activate** command is entered, the dictionary name is placed in *lx_dict_names_p* before calling the Linguistic Tools.

Output Format

The Activate Dictionary function outputs information in the reply area. After the output header displays information from the Linguistic Tools control block, the following information from the reply area is output.

- Dictionary token
- Dictionary type
- Dictionary content
- Dictionary version number
- Dictionary language
- Fully-qualified dictionary filename

Example

```
     ACT us.dic


Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: ACTIVATE        Return Code: LX_RC_OK

Token  Type   Content  Version  Language  Name
-----  -----  -------  -------  --------  -------------------
  1      0     0xE7      23       6011    d:\dicts\us.dic
```

AddMorph

SYNTAX

ADDMorph *dictionary_token_#*

DESCRIPTION

The **AddMorph** command will take both *lx_word_morph_reply_p* and *lx_alias_morph_reply_p* as input. Each lemma in *lx_word_morph_reply_p* will be added to the specified addenda, with the corresponding lemma from *lx_alias_morph_reply_p* included as its morphological alias. The user must execute the **CopyBuff** command to set *lx_alias_morph_reply_p* and *lx_word_morph_reply_p* before calling this function so both areas will contain information.

Input Format

The only input parameter for the **AddMorph** command is the token number of the addenda dictionary to which the information is to be added. The input in the reply areas pointed to by *lx_word_morph_reply_p* and *lx_alias_morph_reply_p* is produced by previous calls to the command.

Output Format

The only output produced by the **AddMorph** command is the header information.

Example

```
gif table
copybuff lx_alias
gif /alias=table rable
copybuff lx_word
addmorph 1

Batch Input File: (command line)   Text Input: (none)
        Function: ADDMORPH        Return Code: LX_RC_OK
```

AddWord

SYNTAX

ADDWord *dictionary_token_# word additional_data*

DESCRIPTION

The **AddWord** command adds specified word(s) to an addenda dictionary. The token number provided with the command specifies the addenda dictionary to which the word(s) should be added.

Input Format

If no input is entered after the dictionary token number, the current data-element list, usually created from an **InFile** command, is used. If no additional data is to be associated with any word(s), the data-element list may be either tagged (/b option of the **InFile** command) or untagged (/s option of the **InFile** command). If additional data is to be stored with any word(s), the data-element list must be in tagged format (/b option of the **InFile** command), using the tags described in "Tagged Input Format — Option /b of the InFile Command" on page 6. In any case, only one word is permitted in each data element (one word per text line), because the **AddWord** command accepts only single-word input. All valid non-duplicate words in the input data-element list will be added to the specified addenda.

If the **InFile** command is not used to create the data-element list, the text input on the command line should specify the word to be added, followed by any additional data which is to be stored with it. The additional data should be in the following tagged format:

```
<data-type> data
```

Note: The space between the data-type indicator and the data is required. Also, the angle brackets (< >) around the data-type are required.

Acceptable data types and their expected data values are:

```
<DATA>      any alphanumeric string – user-defined data
<DBPC>      a list of Double-Byte Parse Codes, separated by commas
<HYPH>      a copy of the input word with its hyphenation points
            indicated – this can be null if the hyphenation points are
            indicated in the input word itself
<MORPH>     a single morphology alias word
<SYN>       a list of synonyms, separated by commas
<ABBR-EOS>  present if the input word is an abbreviation and it can
            appear at the end of the sentence – not present if the input
            word is not an abbreviation, or if the input abbreviation
            cannot occur at the end of a sentence
```

Output Format

The **AddWord** command outputs the data of each data element. After the output header displays information from the Linguistic Tools Control Block, information from the data-element list is output.

Example

```
ADDWORD 3 Goldilocks

Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: ADDWORD         Return Code: LX_RC_OK

      Input  Data  Trailing
RC    Flag   Byte  Delimiter  Length  Data
----  -----  ----  ---------  ------  ----
73    0x00   0x00    0x20       10    Goldilocks
```

ArtCheck

SYNTAX

ARTcheck *text*

DESCRIPTION

The **ArtCheck** command invokes the article-checker function of the Linguistic Tools.

Input Format

If no input text follows the command, the current data-element list is used. If the data-element list was created by an **InFile** command, the input text may be tagged (/b option of the **InFile** command) or untagged (/s or /f option of the **InFile** command).

Note that if the /b option is used and endblanks=0 (the default) in the **:Element** tag and there is no other tag (new-line, new-paragraph, user-data, and so on) to force separation of the text at the end of one data element from the text at the beginning of the next, this text will be concatenated. Using the /s option will also result in text being concatenated from one data element to another. Because the /f option tells the tester to add a new-line element after every data element that it creates, text from one data element will not be concatenated to the text from another.

Output Format

The **ArtCheck** command outputs the data from each data element. The data byte field of each data element will indicate whether the word is correctly used in the present context.

Example

```
ARTCHECK

Batch Input File: TESTBAT.TSC      Text Input: TEXT.TIN
        Function: ARTCHECK        Return Code: LX_RC_OK

       Input    Data    Trailing
 RC    Flag     Byte    Delimiter    Length      Data
----   -----    ----    ---------    ------      ----
 73     0x00    0x00      0x20         4         This
 73     0x00    0x00      0x20         2         is
 73     0x00    0x00      0x20         1         a
 73     0x00    0x00      0x2E         4         test
```

Available Dictionaries

SYNTAX

AVL_dicts *option*

DESCRIPTION

The **AVL** command causes the Linguistic Tools to search the user's system for files that it can identify as dictionaries. It may return either a list of locations on the system where dictionaries were found or a list of all the files that it was able to locate and identify as Linguistic Tools dictionaries.

Input Format

Option

Default or *-f* When the **AVL** command is issued without any options or with -f, it will list information on all the available dictionaries, including dictionary type, content, version, language code, and fully qualified name, and it will return LX_RC_OK.

-p When the **AVL_Dicts** command is issued with an option of -p, it will list the directories in which dictionaries are located and complete with a return code of LX_RC_OK. The list of directory names will be in a format compatible with the search path format specified under the heading "Dictionary Search Path" in *IBM Dictionary and Linguistic Tools Application Programming Interface Description*, SC30-4039.

Output Format

The **AVL** function will output two different kinds of information depending on the option used.

Example - OS/2

```
AVL

 1:   4   37  16   6011  D:\QUIK\QUIKADD.ADD
 2:   4   37  16   6011  D:\NLPEXE\ADTEST1.ADD
 3:   4   37  16   6011  D:\NLPEXE\ADTEST2.ADD
 4:   4   37  16   6011  D:\NLPEXE\ADTEST3.ADD

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK


.AVL -p

D:\QUIK\;D:\NLPEXE\;D:\DICTS\;

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK
```

Example - AIX

```
AVL

 1:   0  231  23   6011  /nlp/dict/us.dic
 2:   4   37  16   6011  /nlp/addn/adtest1.add
 3:   4   37  16   6011  /nlp/addn/adtest2.add
 4:   4   37  16   6011  /nlp/addn/adtest3.add

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK


.AVL -p

/nlp/dict:/nlp/addn

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK
```

Example - Mac

```
AVL

 1:   0  101  21   5991  nlp:dict:aus.dic
 2:   0  245  23   2168  nlp:dict:afrikaan.dic
 3:   0  245  23  14034  nlp:dict:aktueel.dic
 4:   0  229  23  16072  nlp:dict:brasil.dic

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK


.AVL -p

nlp:dict;nlp:dct;nlp:poe24:test

Batch Input File: AVL1.TSC         Text Input: (none)
        Function: AVL_DICTS       Return Code: LX_RC_OK
```

Call

SYNTAX

CALl *testcase.fil*

DESCRIPTION

The **Call** command can be used to invoke a test case from an interactive session of the Linguistic Tester.

Input Format

The **Call** command requires the name of the file that contains a series of Linguistic Tester commands The **Call** command

Output Format

The **Call** command produces no output of its own, but displays whatever output the called test case would have produced.

CB

SYNTAX

CB *fieldname = value*

DESCRIPTION

The **CB** command allows the user to set fields or view field values in the control block. The fieldnames that may be used with this command are listed in Table 5.

Input Format

If **CB** is entered without a field name, the current values of all of the fields described in this section will be output. If a field name is entered without the equals sign and a value, the current value of that field will be output.

When entering values with the **CB** command, spaces are optional around the equals sign.

Table 5 (Page 1 of 3). CB Field-Name Attributes

Field Name	Input Type	Default Value
lx_abbr_tkns	*	0 for all 20 entries
lx_abbr_tkns_ct	Integer	0
lx_add_form_f	Integer	0
lx_add_lang_f	Integer	65535
lx_add_type_f	Integer	0
lx_alg_hyph_f	Boolean	1
lx_alg_only_f	Boolean	0
lx_alias_len	Integer	0
lx_alias_morph_reply_p	Pointer	NULL
lx_alias_p	Pointer	NULL
lx_art_checker_f	Boolean	0
lx_auto_aid_f	Boolean	1
lx_chars_left	Integer	0
lx_comp_aid_f	Boolean	1
lx_comp_position	Integer	0
lx_conf_res	Integer	0
lx_cont_reply_f	Boolean	1
lx_decompnd_further_f	Boolean	0
lx_delivered_out_units	Integer	0
lx_dict_found_in	Integer	0
lx_dict_names_p	*	Pointer to LX_MAXDCTNAME characters, the first of which is NULL
lx_dict_tkns	*	0 for all 20 entries
lx_dict_tkns_ct	Integer	0

Table 5 (Page 2 of 3). CB Field-Name Attributes

Field Name	Input Type	Default Value
lx_distance_meas	Integer	0
lx_elements_ct	Integer	0
lx_elements_p	*	NULL
lx_elt_format	Boolean	0
lx_first_tkn_p	Pointer	NULL
lx_func_code	Integer	0
lx_hFirstToken	Integer	0
lx_lang_code	Integer	6011
lx_lookup_exact_mt_f	Boolean	1
lx_morph_form_id	Integer	0
lx_morph_mask_inp	Integer	0
lx_morph_paradigm	Integer	0
lx_morph_pcode	*	7 NULL characters
lx_morph_pos	Integer	0
lx_next_char_p	Pointer	NULL
lx_next_elemt_p	Pointer	NULL
lx_num_cpg	Integer	850 (500 for host platforms)
lx_out_units	Integer	0
lx_phonetic_f	Boolean	0
lx_pos_f	Boolean	0
lx_pref_hyphen_f	Boolean	1
lx_pref_range	Integer	0
lx_rc	Integer	0
lx_reply_p	*	Pointer to *lx_reply_size* bytes
lx_reply_size	Integer *	5000 (10000 for OS/400)
lx_reply_used	Integer	0
lx_rqst_type	Boolean	0
lx_sent_begin_f	Boolean	0
lx_sentence_f	Boolean	0
lx_serv_area_p	Integer (Pointer for OS/400)	0 (NULL for OS/400)
lx_spell_ver_f	Boolean	0
lx_stopw_f	Boolean	1
lx_stopw_tkns	*	0 for all 20 entries
lx_stopw_tkns_ct	Integer	0
lx_syn_inflect_f	Boolean	1
lx_syn_outtype	Integer	0
lx_syn_pos	Integer	255
lx_synonym_f	Boolean	1

Table 5 (Page 3 of 3). CB Field-Name Attributes

Field Name	Input Type	Default Value
lx_txt_cont_f	Boolean	1
lx_unexpected_fn	Pointer	NULL
lx_unexpected_ln	Integer	0
lx_version_num	Integer	1
lx_word_mod_f	Boolean	1
lx_word_morph_reply_p	Pointer	NULL
lx_wordde_f	Boolean	1
lx_wordex_f	Boolean	1
lx_wordre_f	Boolean	1

Notes:

1. Fields marked with an asterisk (*) require a special input format or contain special processing and are described below.
2. Fields that contain an input type of “Pointer” cannot be changed.

lx_abbr_tkns

The value of the *lx_abbr_tkns* field is automatically updated when an abbreviation list is activated by the **ACT** command or deactivated by a **DEACT** command. The output abbreviation token list is placed in the *lx_abbr_tkns* field in the Linguistic Tools control block. If the *lx_abbr_tkns* field is not set before a call is made to the Linguistic Tools, it will contain all zeros.

lx_dict_names_p

The value of the *lx_dict_names_p* field is automatically updated when a dictionary is activated by the **ACT** command, or when an addenda dictionary is created or saved by the **Create** or **SaveAdd** commands. A pointer to the dictionary name is placed in the *lx_dict_names_p* field in the Linguistic Tools control block.

lx_dict_tkns

The value of the *lx_dict_tkns* field is automatically updated when a dictionary is activated by the **ACT** command or deactivated by a **DEACT** command. The output dictionary token list is placed in the *lx_dict_tkns* field in the Linguistic Tools control block. If the *lx_dict_tkns* field is not set before a call is made to the Linguistic Tools, it will contain all zeros.

lx_elements_p

The value of the *lx_elements_p* field is set when the user enters an **InFile** command to designate an input text file. The pointer to the resulting element list is placed in the *lx_elements_p* field in the Linguistic Tools control block. If the **InFile** command is not entered, the default value of *lx_elements_p* will be Null.

lx_morph_pcode

The *lx_morph_pcode* field contains a string of 7 characters. When the value of this field is modified, the value entered will be padded with spaces to 7 characters, if necessary. If the field is not set, the default value of *lx_morph_pcode* will be 7 spaces.

lx_reply_p

The Linguistic Tester will guarantee that *lx_reply_p* is pointing to an area whose size is equal to *lx_reply_size*. If there is not enough storage available to reserve storage equal in size to *lx_reply_size*, *lx_reply_size* will be set to zero and *lx_reply_p* will be set to **Null**.

lx_reply_size

The value of the *lx_reply_size* defaults to 5000. The Linguistic Tester will guarantee that *lx_reply_p* is pointing to an area whose size is equal to *lx_reply_size*. If there is not enough storage available to reserve storage equal in size to *lx_reply_size*, *lx_reply_size* will be set to zero and *lx_reply_p* will be set to **Null**.

lx_stopw_tkns

The value of the *lx_stopw_tkns* field is automatically updated when a stopword list is activated by the **ACT** command or deactivated by a **DEACT** command. The output stopword token list is placed in the *lx_stopw_tkns* field in the Linguistic Tools control block. If the *lx_stopw_tkns* field is not set before a call is made to the Linguistic Tools, it will contain all zeros.

Output Format

The **CB** command simply outputs the current value of the field that was changed. If an incompatible value or an unrecognized fieldname was entered, an error message will be output. If an invalid value is provided for an integer field, that field will remain unchanged.

Example

```
CB lx_num_cpg = 850

LX_NUM_CPG = 850
```

Complex

SYNTAX

COMPLex *text*

DESCRIPTION

The **Complex** command invokes the Lexical Analysis function of the Linguistic Tools.

Input Format

If any text follows the **Complex** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the data-element list specified by the most recent **InFile** command. This text may be either tagged or untagged. Multiple words on a line (resulting in multiple words in a data element) are acceptable, as the Lexical Analysis function operates on multiple-word input.

Note that if endblanks=0 (the default) in the **:Element** tag and there is no other tag (new-line, new-paragraph, user-data, and so on) to force separation of the text at the end of one data element from the text at the beginning of the next, this text will be concatenated. Using the /s option will also result in text being concatenated from one data element to another. Because the /f option tells the tester to add a new-line element after every data element that it creates, text from one data element will not be concatenated to the text from another.

If text is not provided and the **InFile** command has not yet been executed, the user will be asked to enter a text string.

Output Format

The output format of the complex function is a header followed by token-list output.

Example

```
COMPLEX

Batch Input File: TESTBAT.TSC      Text Input: TEXT.TIN
        Function: COMPLEX         Return Code: LX_END_OF_INPUT


 Token  Right  String Content Begin String String
 Type   Blanks  Flag    Bits   Type Length Value
------- ------ ------ ------- ----- ------ ------
 0x03     0     0x00    0x00   0x00   0
 PROPERTY: FIRSTELEM POINTER VALUE TO "It was"...

 0x01     1     0x01    0x03   0x02   2      It
 PROPERTY: INITCAP

 0x01     1     0x01    0x01   0x01   3      was

 0x01     1     0x01    0x01   0x01   4      just

 0x01     0     0x01    0x01   0x01   5      right

 0x01     2     0x08    0x08   0x08   1      .
 PROPERTY: NOLOOKUP
```

Compound

SYNTAX

COMPOund *word*

DESCRIPTION

The **Compound** function isolates components of an input compound word.

Input Format

If a word follows the **Compound** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the data-element list specified by the most recent **InFile** command.

The input file may be either tagged or untagged, but only one word is permitted per data element, as the **Compound Component Isolation** function accepts only single-word input. Currently only the first data element in the input file is processed.

Output Format

The output format of the **Compound** function is a header followed by information from the reply area.

Example

```
COMPOUND hotdog

Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: COMPOUND       Return Code: LX_RC_OK


 Number of decompositions: 1

Dict.     # of        Elision  Character  Position
Token     components  Flag     Position   Code     Length  Component
--------  ----------  -------  ---------  -------- ------  ---------
  12         2          0         0        0x03      3       "hot"
                        0         3        0x03      3       "dog"
```

Continue

SYNTAX

CONtinue

DESCRIPTION

The **Continue** command causes the tester to issue a continue reply request to the Linguistic Tools. The tester sets the request type to continue reply and passes the current control block to the Linguistic Tools.

Input Format

There are no input parameters for this command.

Output Format

The output format of the **Continue** command depends on the previous function which was invoked. The output format will be the same as that of previous function except that the output header will specify that a continue reply request was issued.

Example

```
CONT


Batch Input File: TESTBAT.LST     Text Input: TEXT.TIN
        Function: CONTINUE       Return Code: LX_RC_OK

                .
                .      (Contents here depend on the previous function)
                .
```

Copy Buffer

SYNTAX

COPybuff *lx_alias*, *lx_word*

DESCRIPTION

The **Copy Buffer** command allows the user to copy the reply area contents into a working buffer. This command will allocate the neccesary space and modify the *lx_alias_morph_reply_p* and *lx_alias_morph_reply_size* or *lx_word_morph_reply_p* and *lx_word_morph_reply_size* control block fields.

Input Format

Input for the **Copy Buffer** command must be either *lx_alias* or *lx_word*. If *lx_alias* is the input parameter, then the reply area is copied to the *lx_alias_morph_reply_p* field of the control block. If *lx_word* is the input parameter, then the reply area is copied to the *lx_word_morph_reply_p* field of the control block.

Output Format

The only output produced by the **CopyBuff** command is the output header.

Example

```
COPYBUFF lx_alias

Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: COPYBUFF       Return Code: LX_RC_OK
```

Create

SYNTAX

CREate *filename1*

DESCRIPTION

The **Create** command allows the application to create an addenda dictionary for use by the Linguistic Tools. The input filename will be placed in the dictionary name list in the control block. If the command is successful, the created addenda will be active and its token will be placed in the reply area. The newly-created addenda must be activated before it can be used by any other Linguistic Tools function.

Input Format

Only a filename is required as input. The extension/file type defaults to .ADB, and the location defaults to the current directory on the workstation platforms.

Output Format

The output format of the **Create** function is a header followed by information from the reply area.

Example

```
CREATE addenda

Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: CREATE          Return Code: LX_RC_OK

Token  Type   Content  Version  Language  Name
-----  -----  -------  -------  --------  -------------------
  1      8     0x25      16       6011    D:\dicts\usr\addenda.adb
```

Current

SYNTAX

CURrent

DESCRIPTION

The **Current** command displays the list of currently active dictionaries.

Input Format

There are no input parameters for the **Current** command.

Output Format

The **Current** command outputs a list of the currently active dictionaries and their associated tokens.

Example

```
CURRENT

Token  Name
-----  ----
  1    us.dic
  2    uk.dic
  3    francais.dic
```

Deactivate

SYNTAX

DEActivate *dictionary_token_#*

DESCRIPTION

The **Deactivate Dictionary** command allows the user to make an active dictionary unavailable for use.

Input Format

The dictionary token that is input will be passed to the Linguistic Tools to be deactivated. More than one dictionary token may be included as input to this function.

Output Format

The only output produced by the **Deactivate** command is the output header.

Example

```
DEACT 2

Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: DEACTIVATE      Return Code: LX_RC_OK
```

Dehyphenate

SYNTAX

DEHyphenate *word*

DESCRIPTION

The **Dehyphenate** command invokes the Dehyphenation function in the Linguistic Tools.

Input Format

If a word follows the **Dehyphenate** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the data-element list specified by the most recent **InFile** command.

The input file may be either tagged or untagged, but only one word is permitted per data element, as the Dehyphenation function accepts only single-word input. Currently only the first data element in the input file is processed.

Output Format

The Dehyphenation function outputs information to the reply area. The length and the string value of the word in the reply area will be output.

Example

```
DEHYPH mother-in-law


Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: DEHYPHENATE    Return Code: LX_RC_OK

Word
Length    Dehyphenated String
------    -------------------
  13      mother-in-law
```

DelFile

SYNTAX

DELfile *filename1*, *filename2*, *filenameN*

DESCRIPTION

The **DelFile** command allows user to delete file(s) from disk.

Input Format

The input filenames must include both file name and extension/file type. A fully-qualified file name must be used if the file is not in the default location. If no location is specified, the file is assumed to be in the current directory on the workstation platforms.

Output Format

The only output produced by the **DelFile** command is the output header.

Example

```
DELFILE testdeu.adb
Batch Input File: delfile.tsc      Text Input: (none)
        Function: DELFILE         Return Code: LX_RC_OK
```

EKW

SYNTAX

EKW *text*

DESCRIPTION

The **EKW** command invokes the **Extract Keyword** function of the Linguistic Tools which extracts key words from text. If text follows the **EKW** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the file designated by the most recent **InFile** command.

Input Format

Text file input may be either tagged or untagged. Multiple-word data elements are acceptable.

Note that if endblanks=0 (the default) in the **:Element** tag and there is no other tag (new-line, new-paragraph, user-data, and so on) to force separation of the text at the end of one data element from the text at the beginning of the next, this text will be concatenated. Using the /s option will also result in text being concatenated from one data element to another. Because the /f option tells the tester to add a new-line element after every data element that it creates, text from one data element will not be concatenated to the text from another.

Output Format

EKW output consists of a header followed by token-list output.

Example

```
EKW

Batch Input File: TESTBAT.TSC     Text Input: TESTDOC.TIN
        Function: EKW            Return Code: LX_RC_OK


Token List:

Element Right  String Content Begin String String
 Type   Blanks  Flag    Bits   Type Length Value
------- ------ ------ ------- ----- ------ ------
 0x03     0     0x00   0x00    0x00    0
 PROPERTY:  CSEN     Sentence #: 0

 0x09     1     0x01   0x03    0x02    2     I
 PROPERTY: CKWD            Type: 1  (hex=  1)
                part-of-speech: 0x04
                        length: 2
                        string: it
                        number: 0
 PROPERTY: CLEM            Type: Lemma property
                part-of-speech: 0x04
                        length: 2
                        string: it
```

```
0x09     1     0x01   0x01    0x01    3      w
PROPERTY: CKWD            Type: 1  (hex=  1)
                part-of-speech: 0x20
                        length: 3
                        string: was
                        number: 1
PROPERTY: CLEM              Type: Lemma property
                part-of-speech: 0x20
                        length: 2
                        string: be

0x09     1     0x01   0x01    0x01    3      t
PROPERTY: CKWD            Type: 1  (hex=  1)
                part-of-speech: 0x08
                        length: 3
                        string: too
                        number: 2
PROPERTY: CLEM              Type: Lemma property
                part-of-speech: 0x08
                        length: 3
                        string: too

0x09     0     0x01   0x01    0x01    3      b
PROPERTY: CKWD            Type: 1 (hex=  1)
                part-of-speech: 0x18
                        length: 3
                        string: big
                        number: 3
PROPERTY: CLEM              Type: Lemma property
                part-of-speech: 0x10
                        length: 3
                        string: big
PROPERTY: CLEM              Type: Lemma property
                part-of-speech: 0x08
                        length: 3
                        string: big
```

Element

SYNTAX

ELEment *text*

DESCRIPTION

The **Element** command creates data elements from the input text. If text is provided with the command, then it will be passed to the Linguistic Tools. Otherwise, it will use the input file designated by the most recent use of the **InFile** command. Output from the **Element** command will be sent to the destination defined by the last use of the **OutFile** command.

Input Format

The Data-Element Creation function accepts either tagged or untagged input. Multiple words in a data element are acceptable.

Output Format

The **Element** function will output the data-element list that is present in the reply area using the data-element list output format. In addition, a flag indicating whether a continue reply may be used will be output.

Example

```
   ELEMENT

Batch Input File: TESTBAT.TSC     Text Input: TESTDOC.TIN
        Function: ELEMENT        Return Code: LX_RC_OK

Continue:  N

      Input     Data       Trailing
RC     Flag     Byte       Delimiter    Length       Data
----   ----     ----       ---------    ------       ----
 0     0x00     0x00         0x20         2          It
 0     0x00     0x00         0x20         3          was
 0     0x00     0x00         0x20         3          too
 0     0x00     0x00         0x2E         3          hot
```

EQT

SYNTAX

EQT *text*

DESCRIPTION

The **EQT** command invokes the **Extract Query Terms** function of the Linguistic Tools which extracts query terms from text. If any text follows the **EQT** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the value last provided by the **InFile** command.

Input Format

Text file input may be either tagged or untagged. Multiple-word data elements are acceptable.

Note that if endblanks=0 (the default) in the **:Element** tag and there is no other tag (new-line, new-paragraph, user-data, and so on) to force separation of the text at the end of one data element from the text at the beginning of the next, this text will be concatenated. Using the /s option will also result in text being concatenated from one data element to another. Because the /f option tells the tester to add a new-line element after every data element that it creates, text from one data element will not be concatenated to the text from another.

Output Format

EQT output consists of a header followed by token-list output.

Example

```
EQT

Batch Input File: TESTBAT.TSC     Text Input: TESTDOC.TIN
        Function: EQT            Return Code: LX_RC_OK


Token List:

Element Right  String Content Begin String String
 Type   Blanks  Flag    Bits   Type Length Value
------- ------ ------ ------- ----- ------ ------
 0x09     1     0x01   0x01    0x01    9      e
 PROPERTY: CKWD            Type: 1  (hex=  1)
                 part-of-speech: 0x08
                         length: 9
                         string: extremely
                         number: 2
                  weight factor: 20
 PROPERTY: CLEM            Type: Lemma property
                 part-of-speech: 0x08
                         length: 9
                         string: extremely
                  weight factor: 15
 PROPERTY: CINF            Type: Inflected Form property
                 part-of-speech: 0x08
```

```
                        length: 9
                        string: extremely
                 weight factor: 10

0x09     0     0x01   0x01    0x01    5      l
PROPERTY: CKWD           Type: 1 (hex=  1)
                part-of-speech: 0x18
                        length: 5
                        string: large
                        number: 3
                 weight factor: 20
PROPERTY: CLEM           Type: Lemma property
                part-of-speech: 0x10
                        length: 5
                        string: large
                 weight factor: 15
PROPERTY: CINF           Type: Inflected Form property
                part-of-speech: 0x10
                        length: 5
                        string: large
                 weight factor: 10
PROPERTY: CINF           Type: Inflected Form property
                part-of-speech: 0x10
                        length: 6
                        string: larger
                 weight factor: 10
PROPERTY: CINF           Type: Inflected Form property
                part-of-speech: 0x10
                        length: 7
                        string: largest
                 weight factor: 10
PROPERTY: CSYN           Type: Synonym property
                part-of-speech: 0x3
                        length: 3
                        string: big
                 weight factor: 6
PROPERTY: CSYN           Type: Synonym property
                part-of-speech: 0x3
                        length: 12
                        string: considerable
                 weight factor: 6
PROPERTY: CSYN           Type: Synonym property
                part-of-speech: 0x3
                        length: 9
                        string: extensive
                 weight factor: 6

            ...  (many more synonyms) ...

PROPERTY: CLEM           Type: Lemma property
                part-of-speech: 0x08
                        length: 5
                        string: large
                 weight factor: 15
PROPERTY: CINF           Type: Inflected Form property
                part-of-speech: 0x10
                        length: 5
                        string: large
```

```
                 weight factor: 10
PROPERTY: CINF            Type: Inflected Form property
                part-of-speech: 0x10
                        length: 6
                        string: larger
                 weight factor: 10
PROPERTY: CINF            Type: Inflected Form property
                part-of-speech: 0x10
                        length: 7
                        string: largest
                 weight factor: 10
```

Execute

SYNTAX

EXEcute

DESCRIPTION

The **Execute** command allows the user to execute a call to the Linguistic Tools without any control block fields being changed beforehand. Before the user calls this command, the control block fields must be set with whatever values are appropriate for the function.

Input Format

There are no input parameters for the **Execute** command.

Output Format

The output produced by the **Execute** command is the same as what would normally be output for the same function.

Example

```
   initialize
   cb lx_func_code = 513
   cb lx_rqst_type = 0
   cb lx_serv_area_p = 1
   EXECUTE

Batch Input File: Testbat.lst      Text Input: (none)
        Function: TERMINATE       Return Code: LX_RC_OK
```

Exit

SYNTAX

EXIt

DESCRIPTION

The **Exit** command allows the user to leave the tester environment.

Input Format

There are no input parameters for the **Exit** command.

Output Format

No output is produced by the **Exit** command.

Example

```
initialize
terminate
EXIT
```

Fuzzy Spell Aid

SYNTAX

FUZzyspellaid *word*

DESCRIPTION

The **FuzzySpellAid** function provides suggested spellings for a given word by gathering from the specified addenda dictionary a list of words which resemble the input word. It differs from regular Spell Aid in the following ways:

- It is not as restrictive as regular Spell Aid in selecting candidate words
- The user can specify the maximum “distance measure” value to use in selecting candidates words (The “distance measure” is a numeric value computed to determine how distant (different) a candidate word is from the input word. A candidate word that is exactly the same as the input word will have a distance measure of zero.)
- It selects candidate words from a specified **addenda** dictionary only (Spell Aid selects from system and addenda dictionaries)
- It returns up to 20 candidate words (Spell Aid returns up to 6)

The candidate words selected from the addenda are ordered so that those which are thought to most closely resemble the input word are put at the top of the list.

Input Format

Only a word is required as input. If an input word is not specified, then the current data-element list is used.

Output Format

The **FuzzySpellAid** command must output data from the reply area. Each dictionary in the active dictionary list has its own header for information. This header contains the dictionary token, the prefix length and the number of candidates for that dictionary. If the dictionary contained any candidates, they are placed after the corresponding dictionary header. The string that is returned in the Linguistic Tester output as “Candidate: ” is the input string.

Example

```
fuzzyspellaid appalachicola

Batch Input File: RFSANACT.TSC     Text Input: (none)
        Function: FUZZYSPELLAID        Return Code: LX_RC_OK

   Candidate: appalachicola

Dictionary
  token        Prefix length    # of candidates
----------     -------------    ---------------
    1               0                3

          appalachicola
      returned distance=0
          ayatollah
      returned distance=4
          class
      returned distance=4
```

GIF

SYNTAX

GIF [/ALIAS=*alias*] [/PCODE=*7_characters*] [/POS=*##*] [/MORPH=*########*] [*word*]

DESCRIPTION

The **GIF** command applies the **Generate-Inflected-Forms** function of the Linguistic Tools. If a word is not provided as input, this function will use the current data-element list as defined by the **InFile** command. The **GIF** command uses the output file designated by the most recent use of the **OutFile** command. When the optional alias word is input, inflected forms for the input word are created from the morphology information of the alias word.

The /PCODE parameter determines the value that is placed in the *lx_morph_pcode* field of the control block. It expects no more than 7 characters as a value. If fewer than 7 characters are input, the remainder of the input field will be filled with spaces. The /POS parameter determines the value that is placed in the *lx_morph_pos* field. It expects a 2-digit hexadecimal value. The /Morph field determines the value that is placed in the *lx_morph_mask_inp* field of the control block. It expects an 8-digit hexadecimal value. If any of these parameters are omitted, their values will default to zero.

Note: The brackets ([]) used in the syntax above indicate an optional input field.

Input Format

If a word follows the **GIF** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the file designated by the most recent **InFile** command.

The input file may be either tagged or untagged, but only one word is permitted per data element, as the **GIF** function accepts only single-word input. Currently only the first data element in the input file is processed.

Output Format

The **GIF** function must output information from the reply area. It will also output the value of the *lx_dict_found_in fields* in the Linguistic Tools control block. The format of the reply area output will be a nested structure that first outputs the length of the word, the word's string value, and the number of paradigms. For every paradigm, a pcode, a part-of-speech, DBCS parse code, paradigm number, and the number of forms for that paradigm will be output. For every form, the tester will output the form id, and a spellings count indicating the number of variants of that form, followed (for each spelling variant) by the length of the form and the form's string value. Finally, the number of morphology masks for that form will be output, with every morphology mask of the form.

Example

```
GIF truck

Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: GIF            Return Code: LX_RC_OK


len: 5   word: "truck"
# of paradigms: 2

    Pcode: EN
    POS (bitmask format): 0x40
    DBPC (DBCS parse code): 0x00
    paradigm: 2
    Number of forms: 2

       form id:  1
           spellings count: 1
           len: 5    form:  "truck"
           mask count: 1
              0x00800000

       form id:  1
           spellings count: 1
           len: 6    form:  "trucks"
           mask count: 1
              0x00400000

    Pcode: EV
    POS (bitmask format): 0x20
    DBPC (DBCS parse code): 0x00
    paradigm: 3
    Number of forms: 5

       form id:  1
           spellings count: 1
           len: 5   form:  "truck"
           mask count: 6
              0x80000000
              0x44800000
              0x42800000
              0x44400000
              0x42400000
              0x41400000

       form id:  2
           spellings count: 1
           len: 6   form:  "trucks"
           mask count: 1
              0x41800000

       form id:  3
           spellings count: 1
           len: 7   form:  "trucked"
           mask count: 6
              0x24800000
              0x22800000
```

```
      0x21800000
      0x24400000
      0x22400000
      0x21400000

form id:  4
    spellings count: 1
    len: 8   form:  "trucking"
    mask count: 1
      0x10000000

form id:  5
    spellings count: 1
    len: 7   form:  "trucked"
    mask count: 1
      0x08000000
```

Grades

SYNTAX

GRAdes *word*

DESCRIPTION

The **Grades** command invokes the **Grade Level Analysis** function of the Linguistic Tools to obtain grade level information for the input word(s). If a word is not provided as input, this function will use the current data-element list as defined by the last **InFile** command. Output will be directed to the output file designated by the last **OutFile** command; if no output file has been specified, results are displayed on the screen.

Input Format

Text file input may be either tagged or untagged. The **Grade Level Analysis** function can operate on either single-word or block format input. If single-word format input has been selected by setting *lx_elt_format* to 1, only one word per input data element is permitted. Otherwise, multiple words per data element are acceptable.

Output Format

The **Grades** command will output data in the data-element list output format. If single-word format is used for input, the data-element list will be accessed through the *lx_elements_p* field of the Linguistic Tools control block. Otherwise, the list will be obtained through the reply area. The data byte field of each data element will contain a hex value representing the grade level of the word. The function also returns an indication of whether a continue reply may be requested.

Example

```
GRADES

Batch Input File: TESTBAT.TSC      Text Input: TEXT.TIN
        Function: GRADES          Return Code: LX_RC_OK


Continue: N

       Input    Data    Trailing
 RC    Flag     Byte    Delimiter    Length      Data
----   -----    ----    ---------    ------      ----
 73     0x00    0x04      0x20         4         This
 73     0x00    0x04      0x20         2         is
 73     0x00    0x04      0x20         2         an
 73     0x00    0x08      0x20         9         obviously
 73     0x00    0x10      0x20        13         idiosyncratic
 73     0x00    0x04      0x2E         7         example
```

Help

SYNTAX

HELp

DESCRIPTION

The **Help** command can be used to display a list of the recognized tester commands with brief descriptions.

Input Format

There are no input parameters for the **Help** command.

Output Format

The **Help** command produces a list of commands with a brief description following each command, as follows:

Activate	Activate Dictionary
AddMorph	Add Morphology to Addenda
AddWord	Add Word to Addenda
ArtCheck	Article Checking
Avl_Dicts	Find/List Available Dictionaries
Call	Call a Tester Command File
CB	Modify/Display Control Block Fields
Complex	Lexical Analysis
Compound	Compound Word Component Isolation
CopyBuff	Copy Reply Area Buffer
Continue	Continue Reply
CopyBuff	Copy Reply Area Buffer
Create	Create Addenda
Current	Display Current Dictionary List
Deactivate	Deactivate Dictionary
Dehyphenate	Dehyphenate
DelFile	Delete File(s) from Disk
EKW	Extract Keywords
Element	Single-Word Data-Element Creation
EQT	Extract Query Terms
Execute	Execute Current Control Block Configuration
Exit	Exit Tester
FuzzySpellAid	Fuzzy Spell Aid
GIF	Generate Inflected Forms
Grades	Grade Level Analysis
Help	Lists Available Tester Commands
Hyphenate	Hyphenate
InFile	Generate Data-Element List from Input File
InflectWord	Inflect Word from Model
Initialize	Initialize
IsDelim	Check if Character is a Delimiter
ListWords	List Addenda Words
Lookup	Lookup Word in Addenda
MID	Morphological Identification
MixedMode	Set/Reset the MixedMode State

OutFile	Redirect Tester Output to File
OutLevel	Specify Level of Tester Output Detail
QryDelim	Query Word Delimiter Table
QryPath	Query Default Dictionary Search Path
Quit	Exit Tester
RegCP	Register Code Page
Remove	Remove Word from Addenda
Reset	Reset Control Block
SaveAdd	Save Addenda
SetDelim	Set Word Delimiter Table
SetExts	Set Default Dictionary Extensions
SetPath	Set Default Dictionary Search Path
Simple	Simple Tokenization
SpellAid	Spell Aid
Synonym	Synonym and Defintion Aid
System	Execute a System Command
Terminate	Terminate
Textseg	Text Segment Identification
Verify	Spell Verify

Hyphenate

SYNTAX

HYPhenate *word*

DESCRIPTION

The **Hyphenate** command applies the **Hyphenate** function in the Linguistic Tools. If a word is not provided as input, this function will use the current data-element list as defined by the **InFile** command. The **Hyphenate** command uses the output file designated by the most recent use of the **OutFile** command.

If necessary, the input word will be passed to the **Hyphenate** function more than one time to allow for all of the hyphenation points to be found. If, at any time, a return code other than LX_RC_OK is returned, the hyphenation function will stop and any results produced up to that point will be output.

Input Format

Text file input may be either tagged or untagged. Because the **Hyphenate** function only accepts single-word format input, only one word is permitted in a data element. The **Hyphenate** command only processes the first data element in the input file.

Output Format

The **Hyphenate** function must output information in the reply area. If the control block fields *lx_chars_left*, *lx_pref_hyphen_f*, and *lx_pref_range* are set so that only one hyphenation point is possible, the output will specify number of characters preceding the point at which the hyphen should be inserted, the return code, the length of the word, and the string into which the hyphen is to be inserted. If all hyphenation points were requested, a value of 255 will be returned for the hyphenation point, and the word length and output string will include embedded hyphens.

Example

```
(all hyphenation points requested)

     HYPH application

     Batch Input File: TESTBAT.TSC     Text Input: (none)
             Function: HYPHENATE      Return Code: LX_SPEL_CHNG

     Hyph    Return   Word     Hyphenated
     Point   Code     Length   String
     -----   ------   ------   --------------------
      255      81       14       ap-pli-ca-tion


(return only one hyphenation point)
     cb lx_chars_left=5
     LX_CHARS_LEFT=5

     HYPH application
```

```
Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: HYPHENATE      Return Code: LX_RC_OK

Hyph    Return   Word     Hyphenated
Point   Code     Length   String
-----   ------   ------   --------------------
 5      73       11       application
```

InFile

SYNTAX

INFIle filename [*/b*] [*/s*] [*/f*]

DESCRIPTION

The **InFile** Command allows the user to designate the name and location of the input text file to be used for either data-element or word-list input. Once the file is designated, any function requiring file input will use the file assigned in this command. Any subsequent invocation of the **InFile** command will overwrite the current input file information with a new value. If the operation is successful, a simple confirmation of the command will be output.

When specifying the input file, a path may optionally be included. If no path is included and the file is not found in the current directory, then the file is searched for in every directory specified by the TESTPATH environment variable.

When the **InFile** command is executed, the tester expects that each line of the input text file ends with a logical end-of-line character. On the PC, this is CR (0x0D) and LF (0x0A). Since these characters are not intended to be part of the data element, the tester strips these characters from the end of the text as EOL characters. Therefore, when writing an input text file, one must use an editor which inserts the CR-LF sequence at the end of each line and the EOF at the end of file. The tester also limits each line of the input text to be no more than 100 bytes including the EOL characters.

The /B, /F, and /S parameter flags allow the user to specify the input format of text input file. If the final flag on the line is /B, tagged input will be assumed. If the final flag is /S or /F, then simple text format will be assumed. If no flag is provided, the tester default is to assume that input is tagged. See “Input from a Text Input File” on page 5 for details of input file format.

Note: The brackets ([]) used in the syntax above indicate an optional input field. No more than one option can be specified at a time.

Input Format

Input for the **InFile** command is a *filename* and a flag indicating the type of file. If no flag is indicated, the default is tagged input (*/b*).

Output Format

No output is created by this command.

Example

```
INF a:testfile.lst /s

(INF a:testfile.lst)
```

Inflect Word from Model

SYNTAX

INFLectword *model_word input_word*

DESCRIPTION

The **Inflect Word** command returns the inflected form or forms of a word based on the model provided by the user.

Input Format

This function does not accept text file input. Both the *model_word* and the *input_word* to be inflected must be entered as arguments of the command.

Output Format

The output of the function consists of the model word, input word, number of forms, and the forms.

Example

```
        INFLECTWORD tables counter

Batch Input File: TESTBAT.TSC        Text Input: (none)
        Function: INFLECTWORD       Return Code: LX_RC_OK


len: 6  model word: "tables"
len: 6  word to inflect: "counter"
   Number of forms: 1
          len: 8  form: "counters"
```

Initialize

SYNTAX

INItialize

DESCRIPTION

The **Initialize** command calls the Linguistic Tools to initialize the conversation.

Input Format

There is no optional input required for the **Initialize** command.

Output Format

The only output produced by the **Initialize** command will be the output header.

Example

```
INIT

Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: INITIALIZE      Return Code: LX_RC_OK
```

IsDelim

SYNTAX

ISDelim [*-wb*] [*prevchar*] [*currchar*] [*nextchar*]

DESCRIPTION

The **IsDelim** command calls the Linguistic Tools function NlpIsDelimiter() to determine if a given character is a word delimiter based on its two surrounding characters. The command expects three characters separated by spaces to follow the command. It accepts an optional flag (-wb) to indicate that the current character marks the beginning of a word.

If the user wishes to pass a space as one of the input characters, the input sequence of a backslash followed by a lower-case “s” (\s) should be input.

Input Format

Input for the **IsDelim** command is an optional parameter *-wb*, which indicates that the current character is the beginning of a new word. If this option is present, it must be the first parameter. The other optional parameters:

- *prevchar* is the text character that immediately precedes the possible delimiter character.
- *currchar* is the text character to be tested as a possible delimiter.
- *nextchar* is the text character that immediately follows the possible delimiter character.

Note: The brackets ([]) used in the syntax above indicate an optional input field.

Output Format

Example

```
 ISDELIM -wb . b \s

Batch Input File: (command line)   Text Input: (none)
        Function: ISDELIM         Return Code: LX_NOT_DELIMITER
```

ListWords

SYNTAX

LIStwords *dictionary_token_#*

DESCRIPTION

The **ListWords** command will list the words in the specified addenda dictionary, up to the amount of space available in the reply area. The function will also output an indication of the additional data stored with each word.

Input Format

Input for the **ListWords** command is a single dictionary token number.

Output Format

The output produced by the **ListWords** command is the header followed by the information from the reply area. The data field before each word indicates the types of data stored with the word:

```
D - Double-Byte Parse Codes
H - Hyphenation data
M - Morphological Alias
S - Synonyms
U - User Data
```

Example

```
LISTW 1

Batch Input File: temp.tsc         Text Input: (none)
        Function: LISTWORDS       Return Code: LX_RC_OK

output units: 5
   Addenda
  Word List
---------
..M..  trae
..MSU  traer
.HM.U  traeria
..M..  trao
..M..  trar
```

Lookup

SYNTAX

LOOkup *word*

DESCRIPTION

The **Lookup** command will search through all active dictionaries in order (according to the dictionary token list) until it finds the word or runs out of active dictionaries. If the word is found in an addenda dictionary, the word and all associated information (morphology, hyphenation and user data) will be output. If the word is found in a system dictionary, only the token of the dictionary in which it was found is output.

Input Format

If no input word is included as a command parameter, input will default to the data-element list specified by the most recent **InFile** command. Text in the input file may be either tagged or untagged. Because the **Lookup** function operates only on single-word input, only one word per data element is permitted. However, the command will process multiple single-word data elements.

Output Format

The output produced by the **Lookup** command is the header followed by the information from the reply area.

Example

```
LOOKUP traeria

Batch Input File: (command line)   Text Input: (none)
        Function: LOOKUP          Return Code: LX_RC_OK


Dictionary   Contents
----------   --------

   1         traeria
             HYPH  = tr ae ri a
             MORPH = locaria
             DATA  = bcdjcbasfkjbsacbasdkcbskdafbsdkafb

             The Linguistic Tester returns
             the data with the tags enclosed by "<" ">" and no
             equals sign.

LOOKUP trattoria

Batch Input File: (command line)   Text Input: (none)
        Function: LOOKUP          Return Code: LX_RC_OK


Dictionary   Contents
----------   --------
```

2	(No additional information.)

The Linguistic Tester returns
the message "(Found in system dictionary.)".

MID

SYNTAX

MID /ALIAS=*alias_word*

DESCRIPTION

The **MID** command applies the **Morphological Identification** function in the Linguistic Tools. If a word is not provided as input, this function will use the current data-element list as defined by the most recent **InFile** command. The **MID** command uses the output file designated by the most recent use of the **OutFile** command. When the optional alias word is input, morphology for the input word is created from the morphology information of the alias word.

Input Format

Text file input may be either tagged or untagged. Input flag data such as for Korean end-of-line processing must be specified by the **Inflag** parameter in the **:Element** tag.

Because **MID** is a single-word function, only one word is permitted in an input data element. However, the command will process multiple single-word data elements.

Output Format

The **MID** function must output information from the reply area. It will also output the value of the *lx_dict_found_in* fields in the Linguistic Tools control block. The format of the reply area output will be a nested structure that first outputs the length of the word, the word's string value, and the number of paradigms. For every paradigm, a pcode, a part-of-speech, a DBCS parse code, and paradigm number will be output, followed by the number of lemmas, and the number of morphology masks for the input form within the paradigm. For every lemma, the length and the string value will be output. Finally, every morphology mask of the input form will be output.

Example

```
MID wound

Batch Input File: TESTBAT.TSC     Text Input: (none)
        Function: MID            Return Code: LX_RC_OK

len: 5   word: "wound"
# of paradigms: 4

    paradigm index: 1
    Pcode:   EN
    POS (bitmask format): 0x40
    DBPC (DBCS parse code): 0x00
    paradigm: 2
    lemma count: 1
       len: 5    lemma:  "wound"
    mask count: 1
              0x00800000
```

```
paradigm index: 2
Pcode:   EV
POS (bitmask format): 0x20
DBPC (DBCS parse code): 0x00
paradigm: 3
lemma count: 1
   len: 5   lemma:  "wound"
mask count: 6
          0x80000000
          0x44800000
          0x42800000
          0x44400000
          0x42400000
          0x41400000

paradigm index: 3
Pcode:   E1J
POS (bitmask format): 0x10
DBPC (DBCS parse code): 0x00
paradigm: 5
lemma count: 1
   len: 5    lemma:  "wound"
mask count: 1
          0x00000000

paradigm index: 4
Pcode:   EV
POS (bitmask format): 0x20
DBPC (DBCS parse code): 0x00
paradigm: 28
lemma count: 1
   len:4    lemma:  "wind"
mask count: 7
          0x24800000
          0x22800000
          0x21800000
          0x24400000
          0x22400000
          0x21400000
          0x00800000
```

MixedMode

SYNTAX

MIXedmode #

DESCRIPTION

The **MixedMode** command indicates whether all uppercase input should be allowed to match mixed case dictionary words. By default, mixed case dictionary words can only be matched with an identical input string — for example, PHD does not match PhD — unless the **MixedMode** flag is on.

Input Format

The only input required is either the number 1 or the number 0. An input of 1 indicates that the **MixedMode** flag is to be set on. An input of 0 indicates that the **MixedMode** flag is to be set off.

Output Format

The **MixedMode** command outputs only the standard tester command header.

Example

```
MIX 1

Batch Input File: TESTBAT.TSC      Text Input: (none)
        Function: MIXEDMODE       Return Code: LX_RC_OK
```

OutFile

SYNTAX

OUTFile *filename*

DESCRIPTION

The **OutFile** command designates the name and location of the output file. If the **OutFile** command is provided with no path or if the **OutFile** command has yet not been given in the current tester session, output will be written to stdout (that is, the screen).

Input Format

Input for the **OutFile** command is a required name indicating a file to receive the output.

Output Format

If the command completes successfully, a simple confirmation of the command will be output.

Example

```
OUT a:Testdata.out

(OUT a:Testdata.out)
```

OutLevel

SYNTAX

OUTLevel #

DESCRIPTION

The **OutLevel** command specifies the level of detail that will be output by all subsequent functions. This command expects a numeric argument to be provided. If this argument is zero, all tester functions will output only the tester output header until another **OutLevel** command is executed. If the argument is not zero, each function will output its default output.

Input Format

Input for the **OutLevel** command is a number indicating the level of output desired. A 0 indicates no output; 1 indicates standard raw output; 2 indicates formatted output.

Output Format

If the command completes successfully, a simple confirmation of the command will be output.

Example

```
OUTLEVEL 0
```

QryDelim

SYNTAX

QRYDelim *size_#*

DESCRIPTION

The **QryDelim** command invokes the NlpQryDelimTable() function. It outputs the current delimiter table used by Linguistic Tools.

The user may optionally provide a size parameter to the **QryDelim** command to indicate how large the table should be that is passed to Linguistic Tools. If no size is provided, a table of 256 bytes will be passed. If a size of zero is provided, a NULL pointer is passed to Linguistic Tools.

Input Format

Input for the **QryDelim** command is an option size of the table to be returned. If no input parameters are present, the entire table of 256 entries is returned. If a size is specified, then only the first size entries of the table are displayed. The size is passed to the Linguistic Tools NlpQryDelimTable as the last parameter.

Output Format

The **QryDelim** command outputs the standard tester command header only if an error was returned by Linguistic Tools. Otherwise, the delimiter table will be output in rows of four, preceded by the current hexadecimal offset. Each element of the delimiter table array will be converted into a string representation of the value if a recognized value is returned. Otherwise, the numeric value of that element will be printed.

Example

```
QRYDELIM 20

 0   LX_DELIM       LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE
 4   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE
 8   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE
 C   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE
10   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE   LX_LOWERCASE
```

QryPath

SYNTAX

QRYPath

DESCRIPTION

The **QryPath** command invokes the NlpQrySearchPath() function. It outputs the current dictionary search path used by Linguistic Tools.

Input Format

There are no input parameters for this command.

Output Format

The **QryPath** command outputs the current dictionary search path followed by the standard tester command header.

Example - OS/2

```
QRYPATH
PATH = C:\;C:\DICTS;M:\LANDICTS;

Batch Input File: (command line)   Text Input: (none)
        Function: QRYPATH         Return Code: LX_RC_OK
```

Example - AIX

```
QRYPATH
PATH = /nlp/dicts:/nlp/addn

Batch Input File: (command line)   Text Input: (none)
        Function: QRYPATH         Return Code: LX_RC_OK
```

Example - Mac

```
QRYPATH
PATH = nlp:dict;nlp:dct;

Batch Input File: (command line)   Text Input: (none)
        Function: QRYPATH         Return Code: LX_RC_OK
```

Quit

SYNTAX

QUIt

DESCRIPTION

The **Quit** command allows the user to leave the tester environment.

Input Format

There are no input parameters for this command.

Output Format

No output is produced by the **Quit** command.

Example

```
initialize
terminate
QUIT
```

RegCP

SYNTAX

REGcp code-page filename

DESCRIPTION

The **RegCP** function invokes the NlpRegisterCodePage() function. It passes a table that is generated from values contained in the input file.

Input Format

The input file should contain 256 hexadecimal numbers to be used in building the table to pass to the Linguistic Tools. More than one number may be present on a line when the numbers are separated by one or more spaces. Any line in the file beginning with an asterisk (*) is interpreted as a comment and does not affect the contents of the input table. A sample file to be used with this command may be found in Appendix C, “Sample Code Page Translate Table” on page 95.

Output Format

The **RegCP** command only outputs the standard tester command header.

Example

```
REGCP 888 trans.tbl

Batch Input File: (command line)   Text Input: (none)
        Function: REGCP           Return Code: LX_RC_OK
```

Remove

SYNTAX

REMove dict-token word

DESCRIPTION

The **Remove** command will allow the user to remove a specific word from an active addenda dictionary. The user must specify the handle of the active addenda dictionary. If no word is given as a command parameter, input will default to the data-element list specified by the most recent **InFile** command.

Input Format

Text file input may be either tagged or untagged. Because **Remove Word** is a single-word function, only one word is permitted in a data element. However, the command will process multiple single-word data elements.

Output Format

The only output produced by the **Remove** command is the standard tester command header.

Example

```
REMOVE 1 traeria

Batch Input File: (command line)   Text Input: (none)
        Function: REMOVE          Return Code: LX_RC_OK
```

Reset

SYNTAX

RESet

DESCRIPTION

The **Reset** command allows the user to reset the values of the Linguistic Tools control block to their initial states. This is useful if the user wishes to isolate one set of tests from preceding tests.

Input Format

There are no input parameters for this command.

Output Format

No output is produced by the **Reset** command.

Example

```
initialize
terminate
RESET
initialize
terminate
quit
```

SaveAdd

SYNTAX

SAVeadd *dictionary_token_#* [*type_flag*] [*format_flag*] [*language_id*] [*filename*]

DESCRIPTION

The **SaveAdd** command will allow the user to save changes made to an active addenda dictionary. The user must specify the handle of the active addenda dictionary. Optionally, the user may save the changes to a different file by supplying a filename. In these instances, the user may also change the type and the format of the resulting file.

Allowed types are:

-A save as a regular addenda

-S save as a stopword addenda

-V save as an abbreviation addenda

Allowed formats are:

-B save as a binary file

-F save as a flat file

Allowed language IDs are:

Any of ID numbers from header files.

Input Format

Input is the dictionary token number of the dictionary to be saved. This token number must be the first input parameter for the command. Additional input parameters are:

-f Save the addenda as a flat file

-b Save the addenda as a binary file

-a Save the addenda as a regular addenda

-s Save the addenda as a stopword addenda

-v Save the addenda as an abbreviation addenda

-l language_identification_#
Save the addenda in the language specified. The language identification numbers are specified in one of the include files delivered with the product. The alias defined for these numbers cannot be used here.

file_name_for_saved_addenda
Save the addenda to a specific file. If this is not specified, then the file name used when the addenda was activated or created is used.

Notes:

1. With the exception of the first parameter (*dictionary_token_#*), the parameters may be specified in any order.
2. The brackets ([]) used in the syntax above indicate an optional input field.

Output Format

The only output produced by the **SaveAdd** command is the standard tester command header.

Example

```
SAVEADD 1 -A -F wordlist
Batch Input File: (command line)   Text Input: (none)
        Function: SAVEADD         Return Code: LX_RC_OK
```

SetDelim

SYNTAX

SETDelim *filename*

DESCRIPTION

The **SetDelim** command invokes the NlpSetDelimTable() function. It builds a table of delimiter values to pass to the function based on values contained in the file specified after the command.

Input Format

The contents of the file indicated by the input parameter should contain 256 words that each match one of the following strings:

LX_LOWERCASE
LX_UPPERCASE
LX_NUMERIC
LX_DELIM
LX_TRAILCHAR
LX_STKNECLASS
LX_STKNFCLASS
LX_STKNGCLASS
LX_STKNHCLASS
LX_STKNICLASS
LX_STKNKCLASS
LX_STKNDBBEG
LX_STKNSHIFT

More than one of these strings may occur on a single line when separated by one or more spaces. Any lines in the file that begin with an asterisk (*) are assumed to be comments and do not affect the contents of the table passed to Linguistic Tools.

The table to pass to the Linguistic Tools is built from the file's values based on the order in which they occur. The first string encountered in the file is placed in the first element of the array and the final value of the file is placed in the 255th position of the array (assuming the first offset of the array is zero). A sample delimiter table for use by the **SetDelim** function may be found in Appendix B, "Sample Delimiter Table File" on page 93.

If the supplied filename is equal to the string NULL (and no file exists in the tester's search path by that name), then the tester will pass a NULL pointer to the Linguistic Tools function. This allows the tester to exercise passing invalid values to **NlpSetDelim()**.

Output Format

The output format of the **SetDelim** command is composed only of the standard tester command header.

Example

```
SETDELIM stkn.tbl

Batch Input File: test.tsc         Text Input: (none)
        Function: SETDELIM        Return Code: LX_RC_OK
```

SetExts

SYNTAX

SETExts *type=extension type=extension* ...

DESCRIPTION

The **SetExts** command allows you to change the default dictionary extensions which Linguistic Tools uses when searching for dictionaries. You may change as many extension defaults as desired at one time by repeating the type/extension pair. The allowable values for type are:

- D17 — Regular system dictionary
- LEG — Legal system dictionary
- MED — Medical system dictionary
- CPT — Computing Terms system dictionary
- ABB — Flat-file abbreviation addenda
- ABR — Binary abbreviation addenda
- ADB — Binary regular addenda
- ADD — Flat-file regular addenda
- STB — Binary stopword addenda
- STW — Flat-file stopword addenda

Any dictionary types not specified will retain their current default extension.

Input Format

Input for the **SetExts** command is a series of *type=extension* indicators, one for each extension you wish to change.

Output Format

The only output produced by the **SetExts** command is the standard tester command header.

Example

```
SETEXTS D17=DCT ADD=WDS
Batch Input File: (command line)   Text Input: (none)
        Function: SETEXTS         Return Code: LX_RC_OK
```

SetPath

SYNTAX

SETPath *path*

DESCRIPTION

The **SetPath** command allows you to change the default dictionary search path which Linguistic Tools uses when searching for dictionaries. The format of the path string must be compatible with the search path format specified in the "Dictionary Search Path" section in the *IBM Dictionary and Linguistic Tools Application Programming Interface Description*, SC30-4039.

Input Format

Input for the **SetPath** command is a set of locations that Linguistic Tools is to use when trying to find an unqualified dictionary name.

Output Format

The only output produced by the **SetPath** command is the standard tester command header.

Example - OS/2

```
SETPATH C:\;C:\DICTS;M:\LANDICTS;
Batch Input File: (command line)   Text Input: (none)
        Function: SETPATH         Return Code: LX_RC_OK
```

Simple

SYNTAX

SIMple *text*

DESCRIPTION

The **Simple** command invokes the simple tokenization function. If a group of text follows the **Simple** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the value last provided by the **InFile** command. If text is not provided and the **InFile** command has not yet been executed, the you will be asked to enter a text string.

Input Format

The **Simple Tokenization** function accepts either tagged or untagged input. Multiple words in a data element are acceptable.

Output Format

The output format of the **Simple** command is a header followed by token-list output.

Example

```
SIMPLE Someone has been sitting in my chair.

Batch Input File: (command line)  Text Input: (none)
        Function: SIMPLE         Return Cdoe: LX_END_OF_INPUT


Token List:

 Token  Right  String Content Begin String String
 Type   Blanks  Flag    Bits   Type Length Value
------- ------ ------ ------- ----- ------ ------
 0x1      1     0x00   0x03    0x02   7    Someone

 0x1      1     0x00   0x01    0x01   3    has

 0x1      1     0x00   0x01    0x01   4    been

 0x1      1     0x00   0x01    0x01   7    sitting

 0x1      1     0x00   0x01    0x01   2    in

 0x1      1     0x00   0x01    0x01   2    my

 0x1      0     0x00   0x01    0x01   5    chair

 0x1      0     0x00   0x08    0x08   1    .

Total number of tokens = 8
```

SpellAid

SYNTAX

SPEllaid *word*

DESCRIPTION

The **SpellAid** command invokes the **SpellAid** function of the Linguistic Tools. If a word is not provided as input, input defaults to the data-element list defined by the most recent **InFile** command. **SpellAid** uses the output file designated by the most recent use of the **OutFile** command.

Input Format

Text file input may be either tagged or untagged. Because **SpellAid** is a single-word function, only one word is permitted in a data element. Currently, the command will process only the first data element in an input file.

Output Format

The **SpellAid** function must output data from the reply area. Each dictionary in the active dictionary list will have its own header for information. This header contains the dictionary token, the prefix length and the number of candidates for that dictionary. If the dictionary contained any **SpellAid** candidates, they will be output after the corresponding dictionary header. The string that is returned in the Linguistic Tester output as “Candidate: ” is the input string.

Example

```
SPELL hous

Batch Input File: (command line)   Text Input: (none)
        Function: SPELLAID        Return Code: LX_RC_OK

   Candidate: hous

Dictionary
  token        Prefix length    # of candidates
----------     -------------    ---------------
    1               0                3

          house
          hours
          hogs

    2               0                1

          haus
```

Synonym

SYNTAX

SYNonym *word*

DESCRIPTION

The **Synonym** command invokes the **Synonym Aid** function of the Linguistic Tools. If a word is not provided as input, then the current data-element list as defined by the last execution of the **InFile** command will be passed to the Linguistic Tools. **Synonym** uses the last output file designated by the most recent use of the **OutFile** command.

Input Format

Text file input may be either tagged or untagged. Because **Synonym Aid** is a single-word function, only one word is permitted in a data element. Currently, the command will process only the first data element in an input file.

Output Format

The **Synonym** function must output data from the reply area. In addition, the number of output units delivered will be output. The output in the reply area will formatted one output field per line. Each field's type length and value will be output.

Example

```
SYN turn


Batch Input File: (command line)   Text Input: (none)
        Function: SYNONYM         Return Code: LX_RC_OK

output units: 6

 Field
         Type     Length       Value
         -----    ------       -----
          1        4           turn
          2        1           1
          6-1     18           To move circularly
          3        0
          7        7           revolve
          7        6           rotate
```

System

SYNTAX

SYStem *system_command_string*

DESCRIPTION

The **System** command passes the specified *system_command_string* to the operating system for execution.

Input Format

The only input for the **System** command is the actual operating system command string to execute. This string is passed to the operating system for execution without any modifications.

Output Format

The output from the **System** command depends on the actual command that was executed. Any output the operating system produced is displayed.

Example

```
SYSTEM DEL JUNK1.TXT
Could Not Find C:\JUNK1.TXT
SYSTEM COPY JUNK.TXT JUNK1.TXT
1 file(s) copied.
```

Terminate

SYNTAX

TERminate

DESCRIPTION

The **Terminate** Command allows the user to invoke the **Terminate Service** function of the Linguistic Tools. The **Terminate** command releases any memory space allocated for use with the Linguistic Tools and writes a record to the Output file stating that the termination was performed and completed.

Input Format

There are no input parameters for this command.

Output Format

The **Terminate** function will produce the output header only.

Example

```
Term

Batch Input File: (command line)   Text Input: (none)
        Function: TERMINATE       Return Code: LX_RC_OK
```

TextSeg

SYNTAX

TEXtseg *text*

DESCRIPTION

The **TextSeg** command causes the Linguistic Tester to invoke the text segment identification function of the Linguistic Tools. If a group of text follows the **TextSeg** command, it is used as input to the Linguistic Tools. Otherwise, input is read from the value last provided by the **InFile** command. If text is not provided and the **InFile** command has not yet been executed, the user will be asked to enter a text string.

Input Format

The **Text Segment Identification** function accepts either tagged or untagged input. Multiple words in a data element are acceptable.

Output Format

The output format of the **TextSeg** function is a header followed by token-list output.

Example

```
TEXTSEG Somebody's been sitting in my chair!


Batch Input File: (command line)   Text Input: (none)
        Function: TEXTSEG         Return Code: LX_END_OF_INPUT

Token list:

 Token  Right  String Content Begin String String
 Type   Blanks  Flag    Bits   Type Length Value
------- ------ ------ ------- ----- ------ ------
 0x3      0     0x00   0x00    0x00   0
 PROPERTY: FIRSTELEM POINTER VALUE TO "Somebo"...

 0x1      1     0x00   0x0b    0x02  10    Somebody's

 0x1      1     0x00   0x01    0x01   4    been

 0x1      1     0x00   0x01    0x01   7    sitting

 0x1      1     0x00   0x01    0x01   2    in

 0x1      1     0x00   0x01    0x01   2    my

 0x1      0     0x00   0x01    0x01   5    chair

 0x1      0     0x00   0x08    0x08   1    !

Total number of tokens = 8
```

Verify

SYNTAX

VERify *word*

DESCRIPTION

The **Verify** command activates the **Verify** function in the Linguistic Tools. It uses the input and output files designated by the most recent use of the **InFile** and **OutFile** commands.

Input Format

If no input text follows the command, it will use the input file designated by the most recent **InFile** command. The input text may be tagged (/b option in the **InFile** command) or untagged (/s or /f option in the **InFile** command).

The **Spell Verify** function can accept either single-word or block format input. If single-word input has been specified by setting *lx_elt_format* to 1, only one word per input data element is permitted. Otherwise, multiple words are acceptable in a data element. In either case, the function will process multiple data elements from an input file.

Note that if endblanks=0 (the default) in the **:Element** tag and there is no other tag (new-line, new-paragraph, user-data, and so on) to force separation of the text at the end of one data element from the text at the beginning of the next, this text will be concatenated. Using the /s option will also result in text being concatenated from one data element to another. Because the /f option tells the tester to add a new-line element after every data element that it creates, text from one data element will not be concatenated to the text from another.

Output Format

The verify function will output data using the data-element list output format. If S format is used for input, the data-element list will be accessed through the *lx_elements_p* field of the Linguistic Tools control block. Otherwise the list will be obtained through the reply area. The return code of each data element will indicate whether the word was found in a dictionary. An indication of whether the reply may be continued is also output.

Example

```
Ver

Batch Input File: (command line)   Text Input: test.tin
        Function: VERIFY          Return Code: LX_RC_OK


Continue : N

      Input  Data  Trailing
RC    Flag   Byte  Delimiter  Length  Data
----  -----  ----  ---------  ------  ----
73    0x00   0x00    0x20        4    This
73    0x00   0x00    0x20        3    one
73    0x00   0x00    0x20        2    is
73    0x00   0x00    0x20        3    too
73    0x00   0x00    0x20        5    small
```

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX
IBM
OS/2
OS/2 Warp
OS/390
OS/400
the IBM logo

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Sun
UNIX

Appendix B. Sample Delimiter Table File

```
* Make the letter 'a' a delimiter
LX_DELIM        LX_DELIM        LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_STKNECLASS
LX_STKNICLASS   LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE    LX_LOWERCASE
LX_LOWERCASE    LX_LOWERCASE    LX_STKNECLASS   LX_STKNKCLASS
LX_NUMERIC      LX_NUMERIC      LX_NUMERIC      LX_NUMERIC
LX_NUMERIC      LX_NUMERIC      LX_NUMERIC      LX_NUMERIC
LX_NUMERIC      LX_NUMERIC      LX_PUNC         LX_PUNC
LX_PUNC         LX_PUNC         LX_PUNC         LX_STKNFCLASS
LX_PUNC         LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_STKNHCLASS   LX_STKNGCLASS   LX_STKNHCLASS
LX_PUNC         LX_STKNHCLASS   LX_STKNHCLASS   LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_PUNC
LX_PUNC         LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_STKNKCLASS
LX_PUNC         LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE    LX_UPPERCASE
LX_UPPERCASE    LX_PUNC         LX_PUNC         LX_UPPERCASE
LX_UPPERCASE    LX_UPPERCASE    LX_STKNICLASS   LX_TRAILCHAR
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_PUNC         LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_STKNHCLASS   LX_DELIM        LX_DELIM
LX_STKNECLASS   LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
LX_DELIM        LX_DELIM        LX_DELIM        LX_STKNHCLASS
LX_DELIM        LX_DELIM        LX_DELIM        LX_DELIM
```

Appendix C. Sample Code Page Translate Table

```
*-------------------------------------------------*
*   SAMPLE EXTERNAL TO INTERNAL TRANSLATE TABLE    *
*-------------------------------------------------*
*-0   -1   -2   -3   -4   -5   -6   -7   -8   -9   -A   -B   -C   -D   -E   -F  */
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xBF 0xFB 0x7F 0x71 0x72 0x73 0x80 0x1B 0x4B 0x9C 0x76 0x4D 0x9B 0x1C 0x3F 0xBE
0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4F 0x50 0x4A 0xED 0x4C 0x75
0x00 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8A 0x8B 0x8C 0x8D 0x8E 0x8F
0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9A 0x74 0x00 0x4E 0x00 0x70
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1A 0x00 0x00 0x00 0xE3 0x00
0xB3 0x31 0x1E 0x28 0x2D 0x23 0x37 0x33 0x29 0x2E 0x24 0x2F 0x2A 0x25 0xAD 0xB7
0x9E 0x38 0xB8 0x2B 0x30 0x26 0x2C 0x27 0x32 0xB0 0xB1 0x3B 0x00 0xBB 0x00 0x00
0x1D 0x1F 0x20 0x21 0x35 0xB5 0x00 0x00 0xBA 0x00 0x00 0x00 0x00 0xB9 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x9D 0xA8 0xA3 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x34 0xB4 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x3C 0xBC 0xA9 0xAE 0xA4 0x39 0x9F 0xAA 0xAF 0x00 0x00 0x00 0x00 0x00 0xA5 0x00
0xA0 0x3A 0xAB 0xA6 0x36 0xB6 0x00 0x3D 0xBD 0xA1 0xAC 0xA7 0x22 0xA2 0x00 0x00
0x70 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x3E 0x00 0x00 0x00 0x00 0x00
```

Printed in U.S.A.

SC30-4040-00