Table of Contents

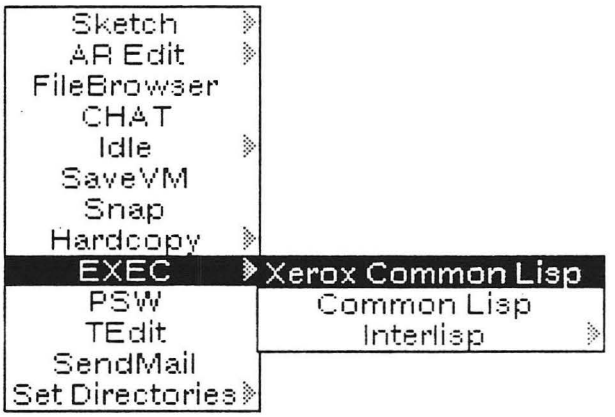
[This page intentionally left blank.]

XEROX

Xerox Lisp

GETTING STARTED IN LYRIC

This document is provided to help you get started using the Lyric release of Xerox Common Lisp. Using a simple example, it presents a step-by-step guide for creating and entering a new package, defining a structure and some functions that operate on the structure, running and debugging the functions, and finally saving and compiling the functions. All of this should be done from an XCL Exec window. Use the right-button background menu as shown below to invoke an XCL Exec window.



The Lyric Example

We're going to define a structure for a WORD; it will have several slots, but the only one we'll use here is a character string that represents the word. We'll then write a few small functions to sort lists of words alphabetically. We're also going to create a package in which we can save the functions for WORD.

All the procedures used in this example are documented in detail in the *Xerox Lyric Release Notes* and the *Xerox Common Lisp Implementation Notes*.

Creating and Entering a New Package

It is easier to create a package and type new code into it, than to create code in an existing package and then move it into a new package. The way Common Lisp DEFSTRUCT treats packages makes this almost mandatory; when a DEFSTRUCT is evaluated, it creates many symbols for structure creation and for accessing functions in the current package, and there is no easy way to collect all these symbols and move them into a new package once they have been used in other code. So, thinking ahead, we create a new package using MAKE-PACKAGE and move into it using IN-PACKAGE. To do this, type the following into your XCL Exec window:

```
(MAKE-PACKAGE "EXAMPLE" :USE '("LISP" "XCL"))
(IN-PACKAGE "EXAMPLE")
```

Defining Common Lisp Structures

It's done with DEFSTRUCT. Type the following into your XCL Exec:

```
(DEFSTRUCT WORD STRING STUFF)
```

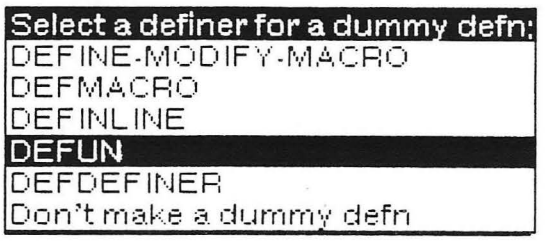
This creates a structure called a WORD with two slots, one named STRING and one named STUFF.

Defining Common Lisp Functions

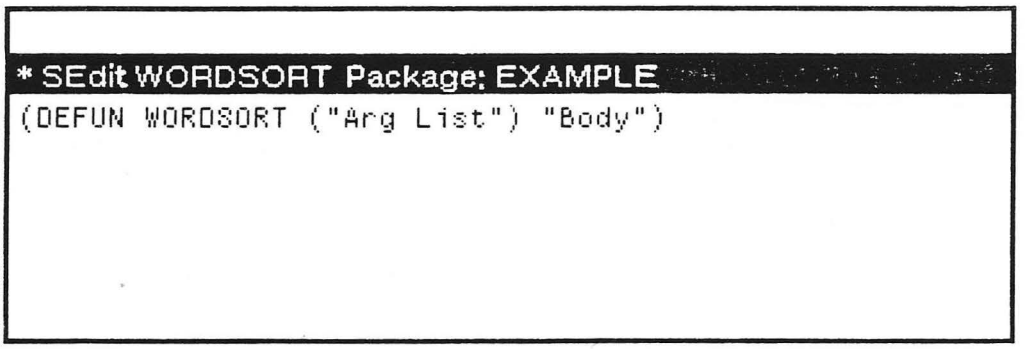
It's done with DEFUN in SEdit. To define functions in SEdit, type the following into your XCL Exec

```
ED(WORDSORT FUNCTIONS)
```

which will pop up a menu of dummy definitions that looks like this



Select the DEFUN entry to invoke an SEdit window that looks like this:



Using SEdit

SEdit should "feel" a lot like TEdit to you; it combines the TEdit style of selection and in-place modification with structure editing.

Like TEdit, SEdit's left mouse button selects a character position, the middle button (or chording left and right on two-button mice) selects a "word" (an atomic object in SEdit), and the right button extends a selection. Extended selections are marked for pending deletion/replacement (if you wish) and displayed with a box around them.

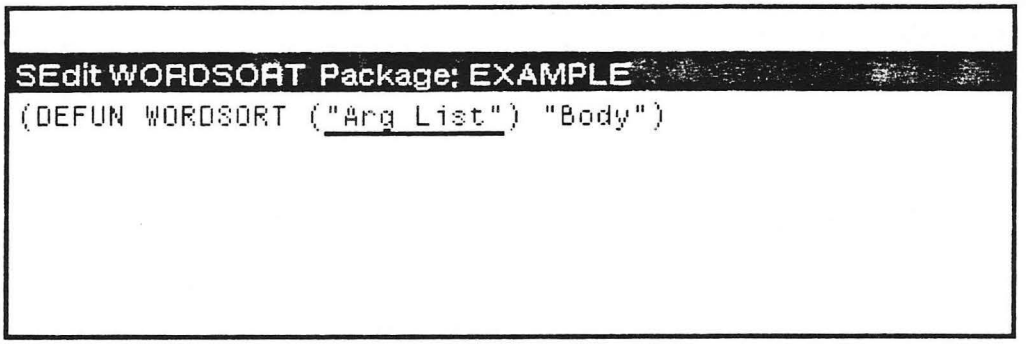
Also, pressing the middle button in the title bar gets you a menu of SEdit operations. One of the most useful of these operations is **Undo**. If you make a mistake using SEdit, put the cursor in the black title bar, press the middle mouse button, and select **Undo**. If you make a number of errors, or wish to start over, select **Abort** from the SEdit command menu.

Unlike TEdit, SEdit has two different types of carets for type-in: a hollow one means that type-in will be added to the current atomic selection, and a filled one means that type-in will create a new object at the caret. Also, repeatedly pressing the left mouse button at the same point causes the current selection to be extended to the next enclosing structure.

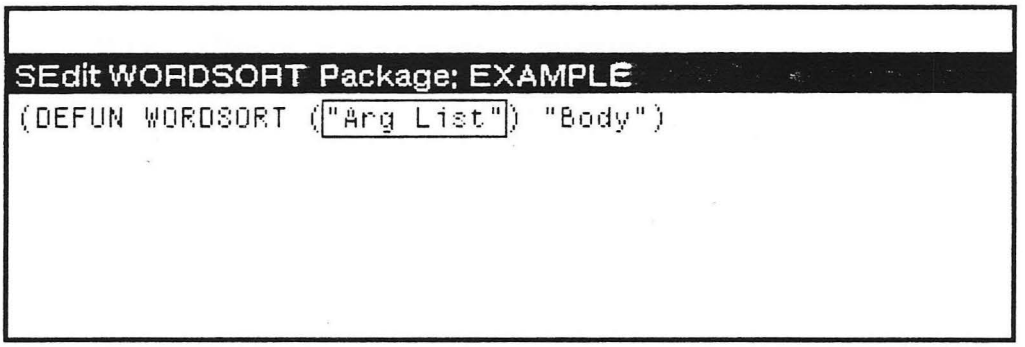
Finally, because SEdit is a structure editor, typing certain characters forces the creation of underlying structure which is then displayed; for instance, typing "(" causes "()" to appear.

For more details on SEdit, look at Appendix B of the Lyric Release Notes. For now, here's what selecting the

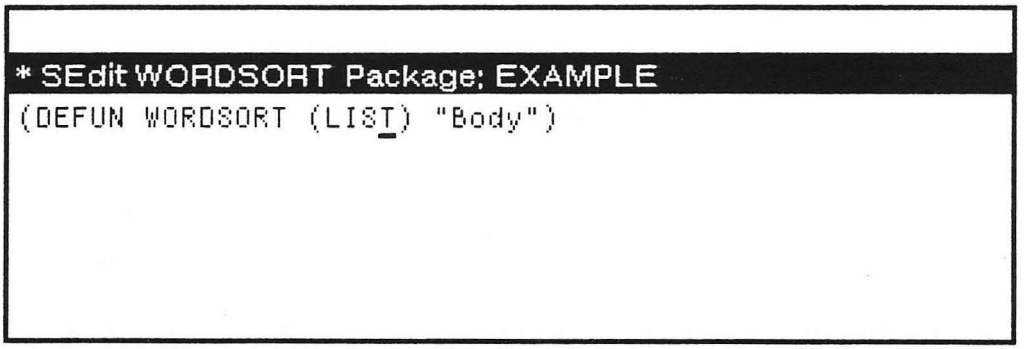
dummy argument list looks like when you use the middle button to select it:



To replace the dummy argument list, put the mouse cursor over the words "Arg List", and press the right button. As shown below, the selection will be enclosed by a box, indicating the area that will be replaced by your type-in.



Type the new argument list "LIST" as shown.



Replacing the dummy body with the code for the function is similarly easy. Notice that SEdit pretty-prints the code as it is typed in; part way through entering the body, it looks like this:

```
* SEdit WORDSORT Package: EXAMPLE
(DEFUN WORDSORT (LIST)
       (IF (OR (NULL LIST)
               (NULL (CDR LIST)))
           LIST (MULTIPLE-VALUE-BIND)))
```

And the final result looks like this (after shaping the window a bit to make the whole function visible):

```
* SEdit WORDSORT Package: EXAMPLE
(DEFUN WORDSORT (LIST)
       (IF (OR (NULL LIST)
               (NULL (CDR LIST)))
           LIST
           (MULTIPLE-VALUE-BIND (LIST1 LIST2)
                  (WORDSPLIT LIST)
                  (WORDMERGE (WORDSORT LIST1)
                             (WORDSORT LIST2)))))
```

The "*" in the title bar means that the original definintion of `WORDSORT` has been modified and not installed yet. You can save `WORDSORT` by closing the SEdit window, by pressing Control-X (which saves the definition but leaves the window open so you can resume editing by clicking in it), or by shrinking the window into an icon that looks like this:

To re-open the window and continue editing, put the mouse cursor over the icon and press the middle mouse button, just as you would with any icon.

To finish the sorter, we need to write `WORDSPLIT` and `WORDMERGE`. The easiest way to open an SEdit window for these is to type "FIX" into the XCL Exec, then just edit the word "WORDSORT" to read

"WORDSPLIT" as shown below. When you've finished writing WORDSPLIT, do the same with WORDMERGE.

```
Exec (XCL)
Select a definer to use for a dummy definition.
WORDSPLIT
36> FIX
36> ED (WORDMERGE FUNCTIONS)
WORDMERGE has no FUNCTIONS definition.
Select a definer to use for a dummy definition.
WORDMERGE
```

The SEdit windows for both are shown below. The comments in them were entered in SEdit by starting with a ";" (entering two or three semicolons starts a comment at different indentations); they are part of the functions' structure like comments in Interlisp code, but unlike them may appear anywhere in Common Lisp forms.

```
* SEdit WORDSPLIT Package: EXAMPLE
(DEFUN WORDSPLIT (LIST)
;;; We know LIST is not empty here, so we only have to check
;;; for the special case of one element before doing the
;;; count-down and split loop.
       (IF (NULL (CDR LIST))
           (VALUES LIST NIL)
           ;; FINDEND moves twice as fast as FINDMIDDLE.
           ;; When FINDEND runs off the end of the list, use
           ;; RPLACD to cut the list in half and return both
           ;; halves; gross, but conses a bit less.
           (DO ((FINDEND (CDDR LIST)
                         (CDDR FINDEND))
                (FINDMIDDLE LIST (CDR FINDMIDDLE)))
               ((NULL FINDEND)
                (SETQ FINDEND (CDR FINDMIDDLE))
                (RPLACD FINDMIDDLE NIL)
                (VALUES LIST FINDEND)))))
```

```
* SEdit WORDMERGE Package: EXAMPLE
(DEFUN WORDMERGE (LIST1 LIST2)
;;; LIST1 is known longer than LIST2 (WORDSPLIT supplies
;;; them that way) so just check whether LIST2 is empty.
       (COND ((NULL LIST2) LIST1)
             ((STRING<= (WORD-STRING (CAR LIST1))
                        (WORD-STRING (CAR LIST2)))
              (CONS (CAR LIST1)
                    (WORDMERGE (CDR LIST1) LIST2)))
             (T (CONS (CAR LIST2)
                      (WORDMERGE LIST1 (CDR LIST2))))))
```

When you have finished writing the definitions for `WORDSPLIT` and `WORDMERGE`, shrink the SEdit windows so that you can easily access them later.

Using the Debugger

To test these functions, we need to build a list of words to sort. This is easily done with an Interlisp `IL:FOR` loop typed into the XCL Exec as shown below:

```
37> (setq wordlist (il:for w il:in '("four" "score" "and"
"seven" "years" "ago") il:collect (make-word :string w)))
(#S(WORD STRING "four" STUFF NIL) #S(WORD STRING "score
" STUFF NIL) #S(WORD STRING "and" STUFF NIL) #S(WORD ST
RING "seven" STUFF NIL) #S(WORD STRING "years" STUFF NI
L) #S(WORD STRING "ago" STUFF NIL))
38>
```

Note the `IL:` package references in front of all the `IL:FOR` keywords.

Now to test `WORDSORT`, we call it with a copy of `WORDLIST`

```
Exec 3 (XCL)

(#S(WORD STRING "four") #S(WORD STRING "score") #S(WORD STRING
"and") #S(WORD STRING "seven") #S(WORD STRING "years") #S(WORD
STRING "ago"))
3/52> (wordsort (copy-list wordlist))
ARG NOT WORD
NIL

3/53>
```

and we discover that it doesn't work, and that it failed so fast that the debugger decided it wasn't worth opening a break window. One way to fix that is to set `IL:HELPFLAG` to `'IL:BREAK!` (which forces the debugger to always break) and to try it again. Type

```
(setq il:helpflag 'il:break!)
```

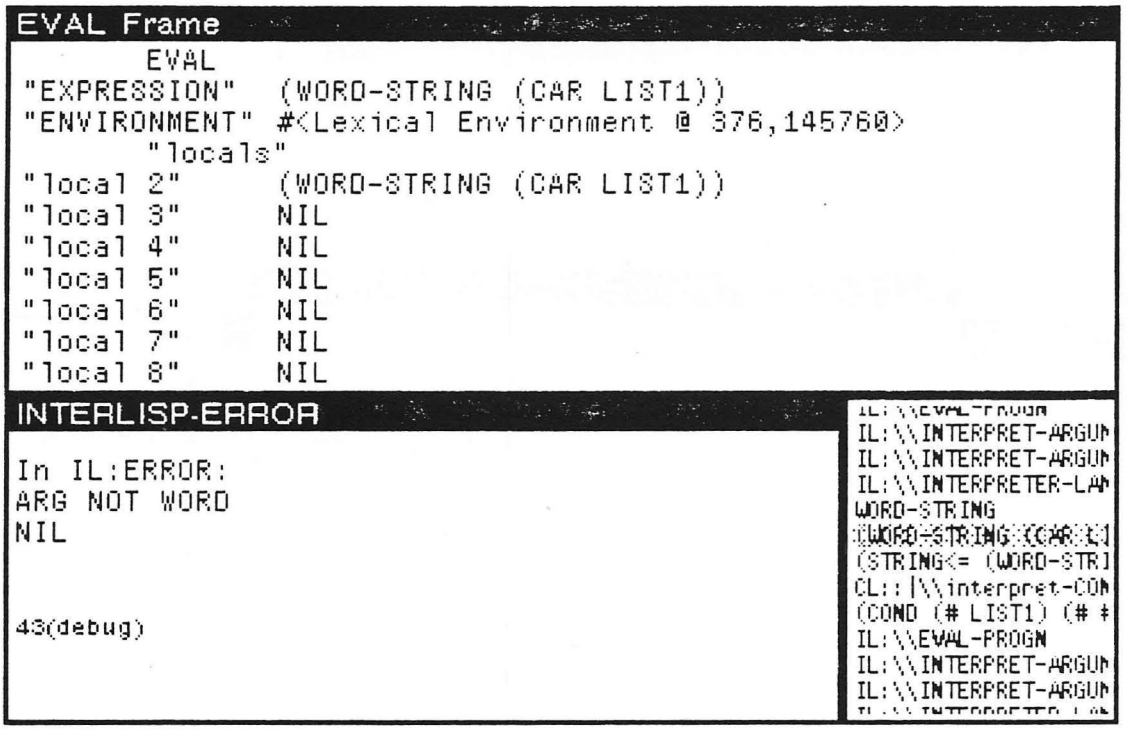
into your XCL Exec, then redo the call to `WORDSORT`.

This gets us a break window, as expected. Now we can poke around on the stack and determine what went wrong.

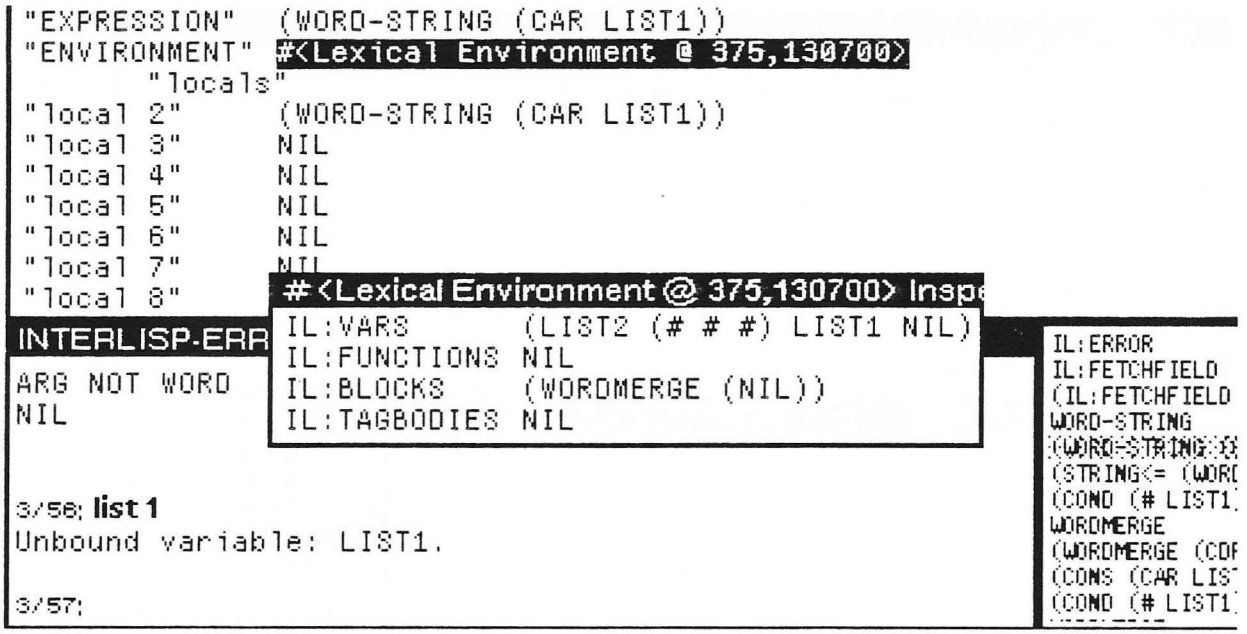
```
INTERLISP-ERROR
In IL:ERROR:
ARG NOT WORD
NIL


3/58:
```

Put the mouse cursor in the black title bar of the break window, and press the middle mouse button. Select **BT** from the pop-up menu to display the stack. Then press the left mouse button to select `(WORD-STRING (CAR LIST1))` in the stack display window to the right of the break window. The resulting frame display looks like this:



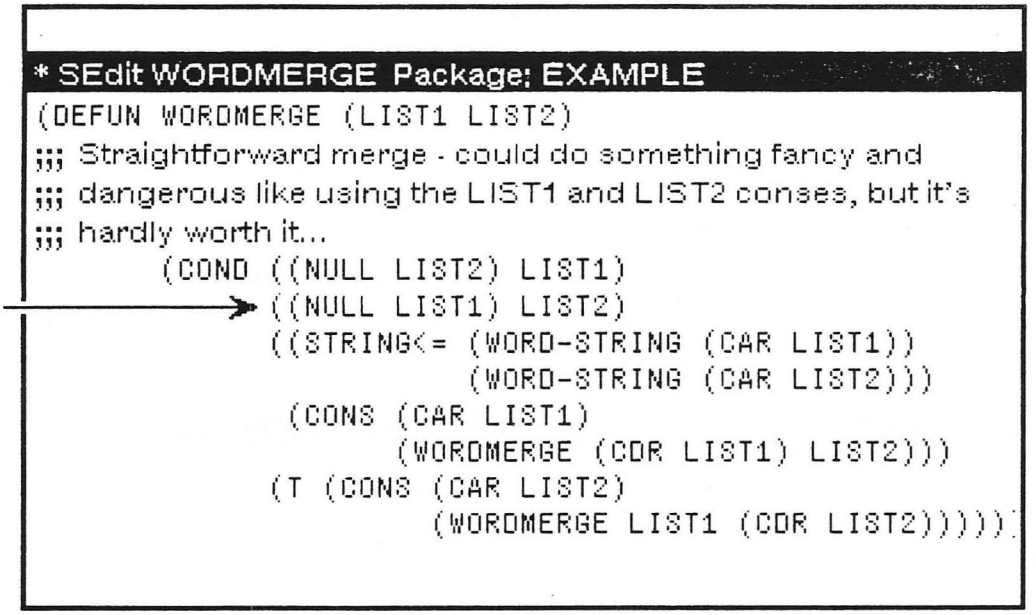
Since `(CAR LIST1)` is `NIL`, it's natural to try to find out the value of `LIST1`. However, `LIST1` is a Common Lisp lexical variable, so it has no dynamic binding and just typing it into the break window doesn't work. To find out the value of a lexical variable, you need to find and inspect the appropriate lexical environment structure. In this case, it happens to be in the current frame display, so select the `#<Lexical Environment @ 357,130700>` using the left mouse button, then press the middle-button and select the **Inspect** option from the command menu.



The variable bindings are stored in property-list form in the `IL:VARS` slot of the lexical environment structure. In this case you can see that `LIST1` is `NIL` immediately, but often it will be necessary to inspect the binding list itself using SEdit to see the bindings in enough detail.

Seeing that `LIST1` is `NIL`, you can look back at `WORDMERGE` and see that it assumes that `LIST1` should not be passed to it as `NIL`. When `WORDMERGE` is called with arguments generated by `WORDSPLIT`, this is correct; however, `WORDMERGE` calls itself recursively and can easily pass `NIL` to itself as `LIST1` The easiest way to fix this is to add a `(NULL LIST1)` clause to the `COND` (and to change the comment appropriately).

Unshrink the `WORDMERGE` SEdit icon, and add the clause as indicated by the arrow.



After fixing the problem and exiting SEdit, you can "↑" out of the break window and try the sort again; this time it works.

```
Exec (XCL)
55> fix wordsort
55> (WORDSORT (COPY-LIST WORDLIST))
(#S(WORD STRING "ago" STUFF NIL) #S(WORD STRING "and" S
TUFF NIL) #S(WORD STRING "four" STUFF NIL) #S(WORD STRI
NG "score" STUFF NIL) #S(WORD STRING "seven" STUFF NIL)
 #S(WORD STRING "years" STUFF NIL))
56>
```

Saving Common Lisp Forms

The File Manager knows how to handle Common Lisp forms in general. In the current example, saying (IL:FILES?) to the Exec gets you the following:

```
Exec 3 (XCL)
the Common Lisp structures: WORD
the Common Lisp functions/macros: WORDMERGE, WORDSPLIT,
        WORDSORT
...to be dumped.  want to say where the above go ? yes
(Common Lisp structures)
WORD  File name: wordstuff
create new file WORDSTUFF ? yes
(Common Lisp functions/macros)
WORDMERGE
```

The creation and specification of new packages is not recorded automatically by the File Manager, though, because packages can have arbitrarily complex inheritance and internal/external symbol structures. The Manager will look for the property IL:MAKEFILE-ENVIRONMENT on the root name for a file to determine the package and readtable to be used to dump the file, so if you want a file to be dumped in its own package, create this property and put a (IL:PROP IL:MAKEFILE-ENVIRONMENT <root-file-name>) in the COMS for the file:

```
Exec 3 (XCL)
WORDSORT  File name: WORDSORT
3/85> (setf (get 'il:wordstuff 'il:makefile-environment)
'(:readtable "XCL" :base 10 :package (defpackage
"EXAMPLE" (:use "LISP" "XCL"))))
(:READTABLE "XCL" :BASE 10 :PACKAGE (DEFPACKAGE "EXAMPLE" (:USE
 "LISP" "XCL")))
3/90> ed(wordstuff files)
Editing FILES definition of WORDSTUFF
```

```
SEdit WORDSTUFFCOMS Package: INTERLISP
((STRUCTURES EXAMPLE::WORD)
 (FUNCTIONS EXAMPLE::WORDMERGE EXAMPLE::WORDSORT
            EXAMPLE::WORDSPLIT)
 (PROP MAKEFILE-ENVIRONMENT WORDSTUFF))
```

Note the EXAMPLE:: package indications in the COMS for the WORDSTUFF file. The DEFPACKAGE expression can be very simple in this example because there are no external symbols in the EXAMPLE package to worry about. Read "Moving Existing Code into a New Package" on page 29 of the *Common Lisp Implementation Notes* and "Reader Environments and the File Manager" in the *Lyric Release Notes* for more on how to handle the more complex cases.

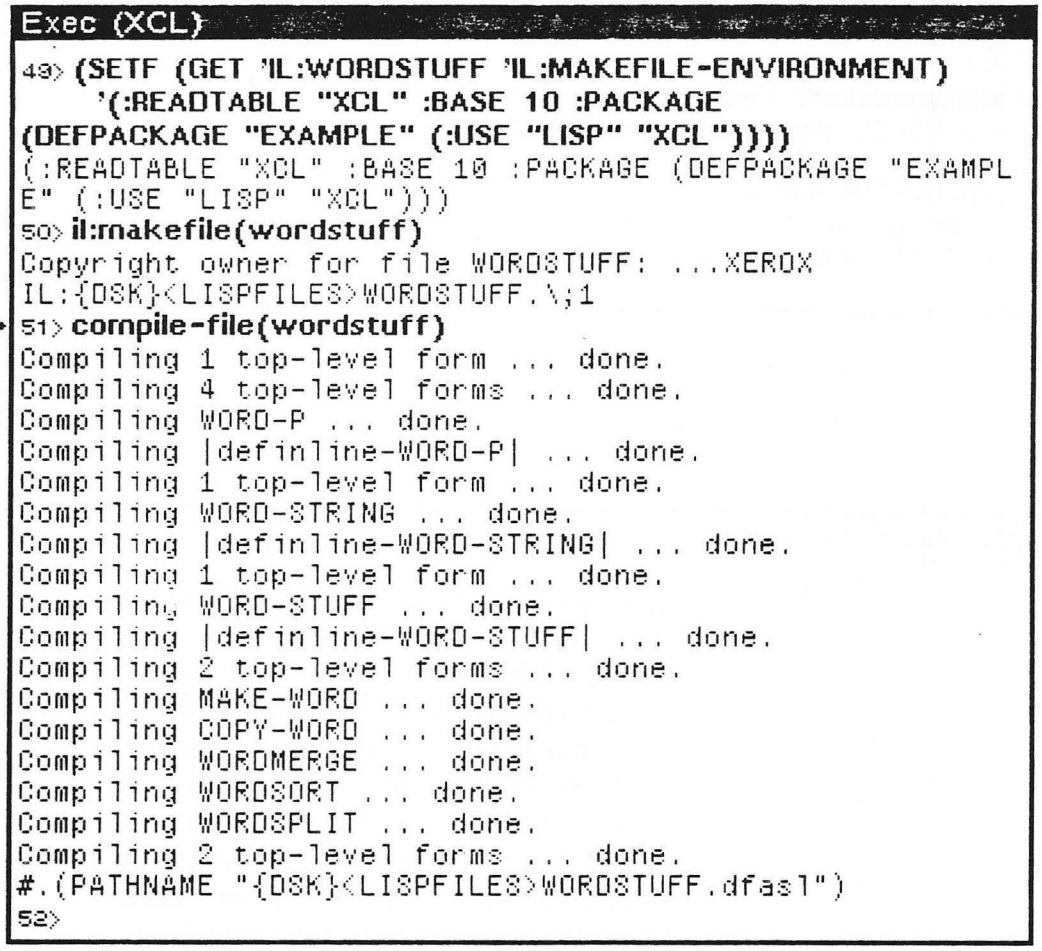
Once the MAKEFILE-ENVIRONMENT property is handled, the file can be IL:MAKEFILEd just like an Interlisp file:

```
Exec 3 (XCL)
(:READTABLE "XCL" :BASE 10 :PACKAGE (DEFPACKAGE "EXAMPLE" (:USE
 "LISP" "XCL")))
3/90> ed(wordstuff files)
Editing FILES definition of WORDSTUFF
WORDSTUFF
3/96> il:makefile(wordstuff)
Copyright owner for file WORDSTUFF: ...XEROX
IL:{DSK}<LISPFILES>WORDSTUFF.;1
3/97>
```

Compiling Common Lisp Files

CL:COMPILE-FILE works for both File Manager files and Common Lisp files (files that are just text). Currently the way it tells them apart is by reading the first character from the file with CL:PEEK-CHAR: if that character is a semi-colon, the file is assumed to be a Common Lisp file, otherwise it is processed as a File Manager file. Except for the extensions documented in the Common Lisp Implementation Notes for Lyric, CL:COMPILE-FILE works as documented in Guy Steele's *Common Lisp: the Language*.

The final step is to compile WORDSTUFF.


Exec (XCL)
49> (SETF (GET 'IL:WORDSTUFF 'IL:MAKEFILE-ENVIRONMENT)
'(:READTABLE "XCL" :BASE 10 :PACKAGE
(DEFPACKAGE "EXAMPLE" (:USE "LISP" "XCL"))))
(:READTABLE "XCL" :BASE 10 :PACKAGE (DEFPACKAGE "EXAMPL
E" (:USE "LISP" "XCL")))
50> il:makefile(wordstuff)
Copyright owner for file WORDSTUFF: ...XEROX
IL:{DSK}<LISPFILES>WORDSTUFF.\;1
51> compile-file(wordstuff)
Compiling 1 top-level form ... done.
Compiling 4 top-level forms ... done.
Compiling WORD-P ... done.
Compiling |definline-WORD-P| ... done.
Compiling 1 top-level form ... done.
Compiling WORD-STRING ... done.
Compiling |definline-WORD-STRING| ... done.
Compiling 1 top-level form ... done.
Compiling WORD-STUFF ... done.
Compiling |definline-WORD-STUFF| ... done.
Compiling 2 top-level forms ... done.
Compiling MAKE-WORD ... done.
Compiling COPY-WORD ... done.
Compiling WORDMERGE ... done.
Compiling WORDSORT ... done.
Compiling WORDSPLIT ... done.
Compiling 2 top-level forms ... done.
#.(PATHNAME "{DSK}<LISPFILES>WORDSTUFF.dfasl")
52>