7. EDITING AND SAVING

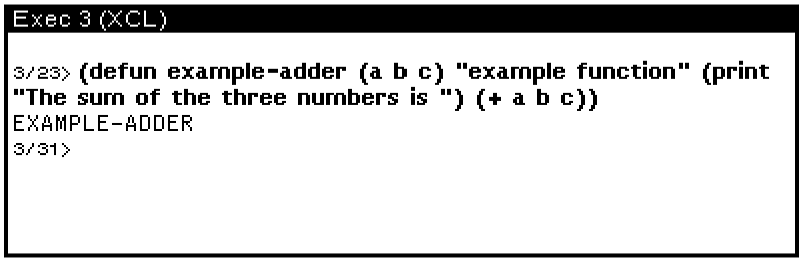
This chapter explains how to define functions, how to edit them, and how to save your work.

Defining Functions

DEFUN can be used to define new functions. The syntax for it is:

```
(DEFUN (<functionname> (<parameter-list><body-of-function>))
```

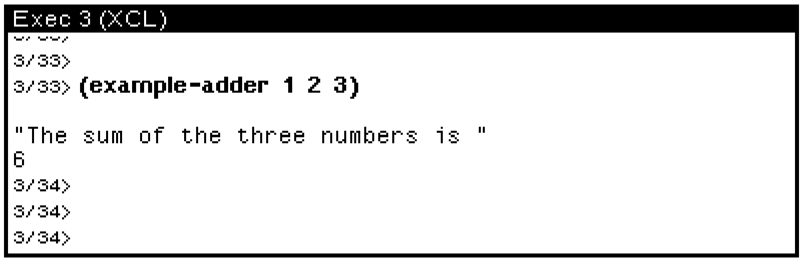
New functions can be created with DEFUN by typing directly into the Executive Window. Once defined, a function is a part of the Medley environment. For example, the function EXAMPLE-ADDER is defined in Figure 7-1.



```
Exec 3 (XCL)
3/23> (defun example-adder (a b c) "example function" (print
"The sum of the three numbers is ") (+ a b c))
EXAMPLE-ADDER
3/31>
```

Figure 7-1. Defining the Function EXAMPLE-ADDER

Now that the function is defined, it can be called from the Executive Window:



```
Exec 3 (XCL)
3/33>
3/33> (example-adder 1 2 3)

"The sum of the three numbers is "
6
3/34>
3/34>
3/34>
```

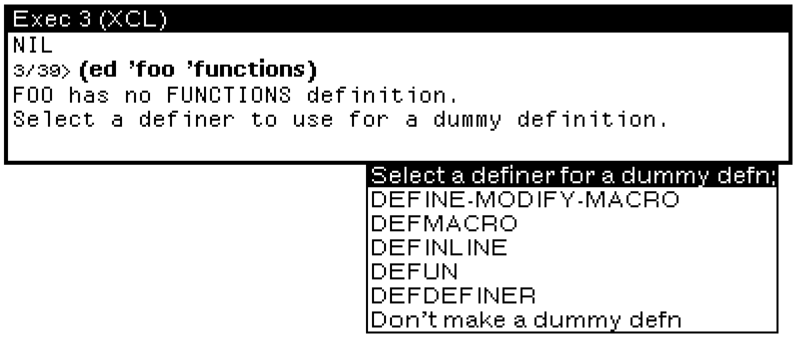
Figure 7-2.. After EXAMPLE-ADDER is defined, it can he executed

The function returns 6, after printing out the message.

Functions can also be defined using the editor DEdit described above. To do this, simply type

(ED *function-name* 'FUNCTIONS)

You will be told that no definition exists for the function, and a menu will pop up asking you what type of function you would like to create:



```
Exec 3 (XCL)
NIL
3/39> (ed 'foo 'functions)
FOO has no FUNCTIONS definition.
Select a definer to use for a dummy definition.
```

Figure 7-3 Selecting a Function Template

Selecting the appropriate type will pop up an editor window with a function template. The use of the editor is explained in the Using the List Structure Editor section below.

Simple Editing in the Executive Window

First, type in an example function to edit:

```
3/41>  (defun your-first-function (a b)
         (if (> a b)
              '(the first is greater)
              '(the second is greater)))
```

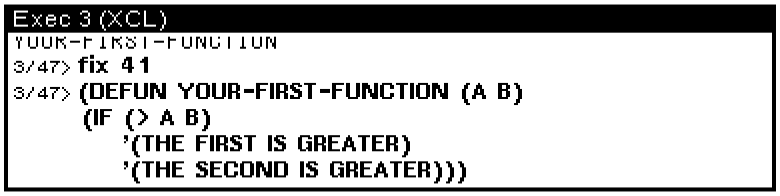
To run the function, type:

```
3/42>  (YOUR-FIRST-FUNCTION 3 5)
(THE SECOND IS GREATER)
```

Now, let's alter this. Type:

```
3/43> FIX 41
```

Note that your original function is redisplayed, and ready to edit. (SeeFigure 7-4.)



```
Exec 3 (XCL)
YOUR-FIRST-FUNCTION
3/47> fix 41
3/47> (DEFUN YOUR-FIRST-FUNCTION (A B)
        (IF (> A B)
           '(THE FIRST IS GREATER)
           '(THE SECOND IS GREATER)))
```

Figure 7-4.   Using FIX to Edit a Fundion

Move the text cursor to the appropriate place in the function by positioning the mouse cursor and pressing the left mouse button.

Delete text by moving the caret to the beginning of the section to be deleted. Hold the right mouse button down and move the mouse cursor over the text. All of the highlighted text between the caret and mouse cursor is deleted when you release the right mouse button.

If you make a mistake, deletions can be undone. Press the UNDO key on the keypad to the left of the keyboard.

Now change `GREATER` to `BIGGER`:

1. Position the mouse cursor on the `G` of `GREATER`, and click the left mouse button. The text cursor is now where the mouse cursor is.
2. Next, press the right mouse button and hold it down. Notice that if you move the mouse cursor around, it will blacken the characters from the text cursor to the mouse cursor. Move the mouse so that the word "`GREATER`" is highlighted.
3. Release the right mouse button and `GREATER` is deleted.
4. Without moving the cursor, type in `BIGGER`.
5. There are two ways to end the editing session and run the function. One is to type Control-**X**. (Hold the Control key down, and type **X**.) Another is to move the text cursor to the end of the line and crø In both cases, the function has been edited!

Try the new version of the function by typing:

```
3/48> (YOUR-FIRST-FUNCTION 8 9)
(THE SECOND IS BIGGER)
```

and get the new result, or you can type:

```
3/49> REDO 42
(THE SECOND IS BIGGER)
```

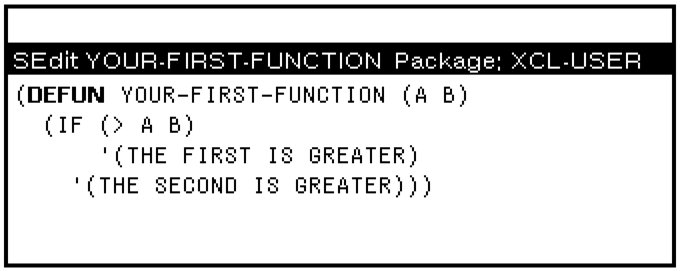
Using the List Structure Editor

If the function you want to edit is not readily available (i.e. the function is not in the Executive Window, and you can’t remember the history list number, or you simply have a lot of editing), use the List Structure Editor, often called SEdit. This editor is evoked with a call to `ED`:

```
81←(ED 'YOUR-FIRST-FUNCTION 'FUNCTIONS)
```

Your function will be displayed in an edit window, as in Figure 7-5.

If there is no edit window on the screen, you will be prompted to create a window. As before, hold the leff mouse button down, move the mouse until it forms a rectangle of an acceptable size and shape, then release the button. Your function definition will automatically appear in this edit window.

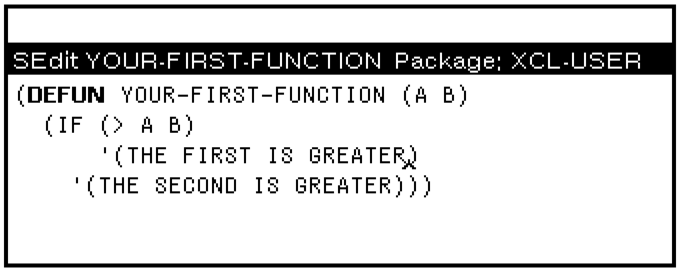


```
(DEFUN YOUR-FIRST-FUNCTION (A B)
  (IF (> A B)
      '(THE FIRST IS GREATER)
    '(THE SECOND IS GREATER)))
```

Figure 7-5. An Edit Window

Many changes are easily done with the structure editor. Notice that by pressing the left mouse button you can place the caret in position, and by pressing the middle mouse button you can select atoms or s-expressions. Repeated pressing of the middle button selects bigger pieces of text.

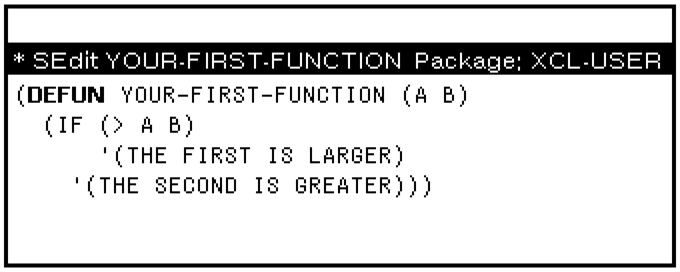
To add an expression that does not appear in the edit window (i.e., it cannot simply be underlined), place the caret at the insertion point and type it in.. For example, to replace the first GREATER with LARGER, place the caret to the left of GREATER, as shown in Figure 7-6.



```
SEdit YOUR-FIRST-FUNCTION  Package: XCL-USER
(DEFUN YOUR-FIRST-FUNCTION (A B)
  (IF (> A B)
      '(THE FIRST IS GREATER)
    '(THE SECOND IS GREATER)))
```

Figure 7-6. Caret Placement Prior to Changing GREATER with LARGER

Now press the DELETE key seven times, and type in LARGER. The window now looks like this:



```
* SEdit YOUR-FIRST-FUNCTION  Package: XCL-USER
(DEFUN YOUR-FIRST-FUNCTION (A B)
  (IF (> A B)
      '(THE FIRST IS LARGER)
    '(THE SECOND IS GREATER)))
```

Figure 7-7. GREATER Changed to LARGER

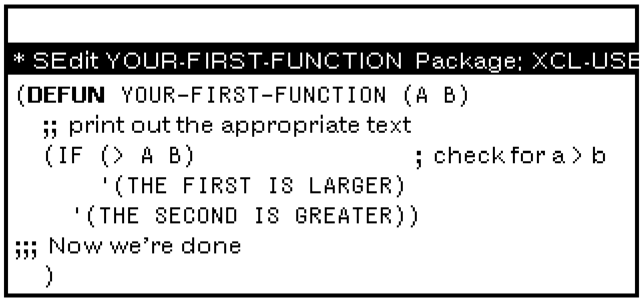
Notice the asterisk in the left edge of the title bar of the window. This designates that the function has be changed. Now exit the edit session by typing Control-X, and the function will be redifined.

Commenting Functions

Text can be marked as a comment by typing a semi-colon before the text of the comment.

```
        ; This is the form of a comment
```

Inside an editor window, the comment will be printed in a different font and may be moved to the far right of the code. SEdit is familiar with the Common Lisp convention of single comments being on the far right, double comments being justified with the function level, and triple comments being on the far left, as is shown in Figure 7-8.



```
* SEdit YOUR-FIRST-FUNCTION  Package: XCL-USE
(DEFUN YOUR-FIRST-FUNCTION (A B)
  ;; print out the appropriate text
  (IF (> A B)                ; check for a > b
      '(THE FIRST IS LARGER)
    '(THE SECOND IS GREATER))
;;; Now we're done
  )
```

Figure 7-8. Placement of Comments

There are other editor commands which can be very useful. To learn about them, read Appendix B of the *Release Notes*.

File Functions and Variables: How to See and Save Them

With Medley, all work is done inside the Lisp environment. There is no operating system or command level other than the Executive Window. All functions and data structures are defined and edited using normal Lisp commands. This sertion describes tools in the Medley environment that will keep track of any changes that you make in the environment that you have not yet saved on files, such as defining new functions, changing the values of variables, or adding new variables. And it then has you save the changes in a file you specify. All of these functions are in the `INTERLISP` (`IL:`) package.

File Variables

Certain system-defined global variables are used by the file package to keep track of the environment as it stands. You can get system information by checking the values of these variables. Two important variables follow.

- `FILELST` evaluates to a list, all files that yoU have loaded into the Medley environment.
- *filename*`COMS` (Each file loaded into the Lisp environment has associated with it a global variable, whose name is formed by appending `COMS` to the end of the filename.) This variable evaluates to a list of all the functions, variables, bitmaps, windows, and soon, that are stored on that particular file.

  For example, if you type:

  ```
      MYFILECOMS
  ```

  the system will respond with something like:

  ```
      ((FNS YOUR-FIRST-FUNCTION  )
       VARS))
  ```

Saving Interlisp-D on Files

The functions `(FILES?)` and `(MAKEFILE '`*filename*`)` are useful when it is time to save function, variables, windows, bitmaps, records and whatever else to files.

`(FILES?)` displays a list of variables that have values and are not already a part of any file, and then the functions that are not already part of any file.

Type:

```
    (FILES?)
```

the system will respond with something like:

```
    the variables: MY.VARIABLE CURRENT.TURTLE...to be
    dumped
```

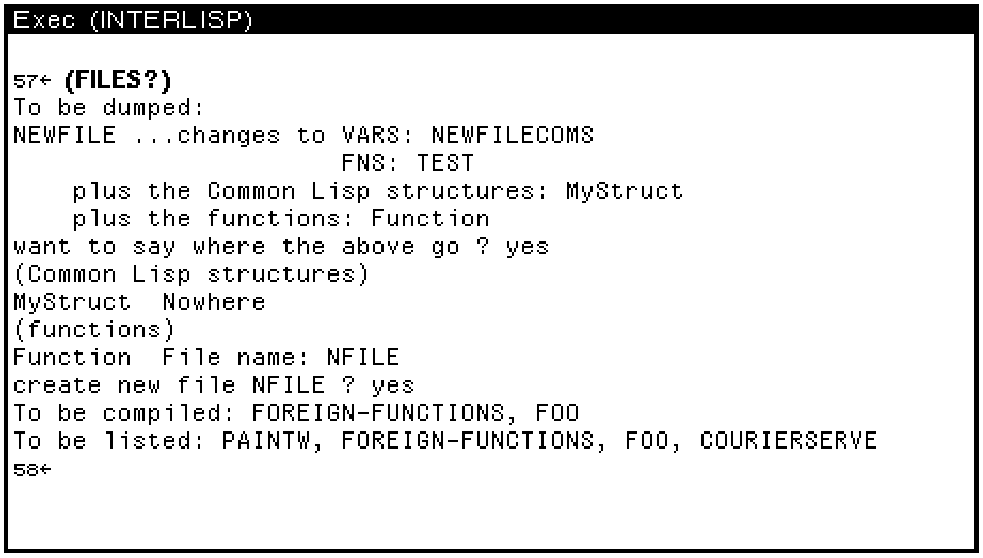
```
the functions: RIGHT LEFT FORWARD BACKWARD
CLEAR-SCREEN...to be dumped

want to say where the above go?
```

If you type Y, the system will prompt with each item. There are three options:

1. To save the item, type the filename (unquoted) of the file where the item should be placed. (This can be a brand new file or an existing file.)
2. To skip the item, without removing it from consideration the next time (FILES?) is called, type crø This will allow you to postpone the decision about where to save the item.
3. If the item should not be saved at all, type]. Nowhere will appear after the item.

Part of an example interaction is shown in the following figure:



```
Exec (INTERLISP)

57← (FILES?)
To be dumped:
NEWFILE ...changes to VARS: NEWFILECOMS
                      FNS: TEST
    plus the Common Lisp structures: MyStruct
    plus the functions: Function
want to say where the above go ? yes
(Common Lisp structures)
MyStruct  Nowhere
(functions)
Function  File name: NFILE
create new file NFILE ? yes
To be compiled: FOREIGN-FUNCTIONS, FOO
To be listed: PAINTW, FOREIGN-FUNCTIONS, FOO, COURIERSERVE
58←
```

Figure 7-9. Part of an interaction using the function FILES?

(FILES?) assembles the items by adding them to the appropriate file's COMS variable (see the File Variables section above). (FILES?) does NOT write the file to secondary storage (disks or floppies). It only upclates the global variables discussed in the File Variables section above.

(MAKEFILE 'filename)

actually writes the file to secondary storage.

Type:

(MAKEFILE 'MY.FILE.NAME)

and the system will create the file. The function returns the full name of the file created. (i.e. {DSK}MY.FlLE.NAME.; 1).

Files written to `{DSK}` are permanent files. They can be removed only by the user deleting them or by reformatting the disk.

Other file manipulation functions can be found in Chapter 4.