Second Group

Date: 19 Dec 91 18:11 PST (Thursday)
Posted-Date: 19 Dec 91 18:19 PST
From: John Sybalsky:PARC:Xerox
Subject: more primer files.
To: porter:mv:envos

>>CoveringMessage<<

----- Begin Forwarded Messages -----

Date: 19 Dec 91 15:28 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.152817pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11662>; Thu, 19 Dec 1991 15:28:23 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:28:17 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

F- 14. BREAKPACliGE

The Break Package is a part of Interlisp that makes debugging your programs much easier.

14.1 Break WindoNT

A break is a function either called by the programmer or by the system when an error has occurred. A separate window opens for each break. This window works much like the Interlisp-D Executive Window, except for extra menus unique to a break window. Inside a break window, you can examine variables, look at the call stack at the time of the break, or call the editor. Each successive break opens a new window, where you can execute functions without disturbing the original system stack. These windows disappear when you resolve the break and return to a higher level.

14.2 Break Package Example

This example illustrates the basic break package functions. A more complete explanation of the breaking functions, and the break package will follow.

The correct definition of FAGTORIAL is:
(DEFIKEQ (FMT0RIAL (xj
then 1

(iføf5ø (ITIES x (f~ToRIAL (sue, xj

To demonstrate the break package, we have edited in an error: DUffKY in the IF statement is an unbound atom, it lacks a value.
(D—FIKEQ (F~T0RIAL (xj
then ~

(if~[~~ (ITIKES x (FACTORIAL ~suei xj

The evaluated function
(F~T0RI~ 4)

should return 24, but the above function has an error. DUMMY

is an unbound atom, an atom without an assigned value, so Lisp will "break". A break window appears (Figure 14.1), that has all the functionality of the typing Interlisp-D expressions into the lnterlis~D executive window (The top level), in addition to the break menu functions. Each consecutive break will move to another level "down".

BREAK PACKAGE 141

BREAK PACKAGE EXAMPLE

51+(PP Fllu'T&RIAL)
cFACTORlAL

[LA'NBOR ! '.j "łrOMn—NT~ł
(if (EROP '~
i,,ien Dummy

6Jil (lTIflEc A !FR~TORIAL !.UB1 :~j;
!FACTCPIALj
5?(FALTORIAL 4,1

DUMMY (in FAi',TORlALJ in =ERDP P1!t4flY
only br'okøon!

Figuro I..l. Break window

Move the mouse cursor into the break window and hold down the middle mouse button. The Break Menu will appear. Choose BT. Another menu, called the stack menu, will appear beside the break window. Choosing stack items from this menu will display another window. This window displays the function's local variable bindings, or values. (See Figure 14.2) This new window, titled FACTORlAL Frame, is an inspector window. (See inspector Chapter 32).
Sr

fau'TUR[AL

EP.PoM5ET
fiRE&1

UNBOUND ATOM LflQ
DUMMY (in fAcTORIAL) in \(ZEROP x) DUMMY) cob ø
FLøiDRI~

(DUMMY broken) cob
FkWRl~
L.OB

F,c~RI~
L'4M0

Figun 14.3. Back Yraco of trio 5ystem Stack

From the break window, you can call the editor for the function FACTORIAL by typing
(OF F~15IL)

Underline X. Choose EVAL from the zditor menu. The value of X at the time of thff break will appear in the edit buffer below tho editor window. Any list or atom can be evaluated in this way (See Figure 14.3.)

14.1 lRF~PACMA'GF

BREAK PACKAGE EXAMPLE

UNBOUND ATOM

ø DUMMY (in FACTORIAL ~ (ITIKES x \øfASTORIAL ~SUB1 X))))) Replace
switch

( )

ø (DUMMY broken) ()cUt

OF FAL'TORIAL) Undo

Find
Swop
Reprint
Edit

EatCam
Break
E~a1
E.t

Figure 14.3. Editing from the Break Window

Replace the unbound atom DUffNY with 1 ø Exit the editor with
the EXIT command on the editor menu.

The function is fixed, and you can restart it from the last call on
the stack (It does not have to be started again from the Top
Level) To begin again from the last call on the stack, choose the
last (top) FACTOR1AL call in the BT menu. Select REVERT from
the middle button break window, or type it into the window.
TThe break window will close, and a new one will appear with
the message: FACTORlAL broken.

To start execution with this last call to FACTORIAL, choose OK
from the middle button break menu. The break window will
disappear, and the correct answer, 24, will be returned to the top
level.

14.3 _ Ways to _ Stop _ Execution _ from the _ Keyboard, called _ "Breaking _ Lisp"
There are ways you can stop execution from the keyboard. They
differ in terms of how much of the current operating state is
saved:

Control-G provides you with a menu of processes to Interrupt. Your process
will usually be ø' EXEC". Choose it to break your process. A break
window will then appear.

Control-B causes your function to break, saves the stack, then displays a
break window with all the usual break functions.
For information on other interrupt characcers, see the Interlisp
Reference Manual, volume 111, page 30.1.

8REAKPAcKAG— 14.3
I

PROGRAMMING BREAKS AND DEBUGGlNG CODE

14.4 Programming Breaks and Debugging Code
PrOgramming breaks are put into code to cause a break when

that section of code is executed. This is very useful for debugging code. There are 2 basic ways to set prOgramming breaks:

(BREAK functionna:e) This function call made at the tOp level will cause a break at the start of the execution of "functionname". This is helpful in checking the values of parameters given to the function.

Setting a break in the editor Take the function that you want tO break into the editor. Underline the expression that should break before it is evaluated. Choose BREAK on the editor command menu. Exit the editor. The function will break at this spot when it is executed.

Once the function is broken, an effective way tO use the break window for debugging is to put it into the editor window. (See Section 14.2, Page 14.2.) All the local bindings still exist, so you can use the editor's EVAL command to evaluate lists, variables, and expressions individually. Just underline the item in the usual way (move the mouse to the word or parenthesis and press the leff mouse button), then choose EVAL from the command menu. (See Section 14.2 for more detail.)

Both kinds of programmed breaks can be undone using the (UNBREAK) function. Type
(~KBRDF functionnm)

Calling (UNBREAK) without specifying a function name will unbreak all broken functions.

14.5 Break Menu

Move the mouse cursor into the break window. Hold the middle button down, and a new menu will pop up, like the one in Figure 14.4.

OK
BT
BY!
"a

f~ure 14.& Thø middle bUtton menu in the Break window
Five of the selection& are particularly important when just starting to use lnterlis~D:

8T Sack Trace displays the stack in a menu beside the break window. Back Trace is a very powerful debugging t00l. Each function call is placed on tho stack and removed when the execution of that function is complete. Choosing an item on thø stack will open another window displaying that item's local

1(. 8~xpAcl:AGE

E~

BREAK MENU

voriobles and their bindings. This is on inspector window thit offers all the power of the inspector. (For details, see the section on the Inspector, Chapter 32).

? Sefore you use this menu option, display the stack by choosing 8T from this menu, and choose a function from it. Now, choose 7: It will display the current values of the arguments to the

function that has been chosen from the stack.

~ Move back to the previous break window, or if there is no other break window, back to the top level, the InterlispøD Executive Window.

REVERT Move the point of execution back to a specified function call before the error. The function to revert back to is, by default, the last function call before the break. If, however, a different function call is chosen on the BT menu, revert will go back to the start of this function and open a new break window. The items on the stack above the new starting place will no longer exist. This is used in the tutorial example. (See Section 14.2, Page 14.1.)
OK Continue execution from the point of the break. This is useful if you have a simple error, i.e. an unbound variable or a nonnumeric argument to an arithmetic function. Reset the variable in the break window, then select OK. (See Section 14.2.)
(Note: In addition to being available on the middle button menu of the break window, all of these functions can be typed directly into the window. Only ST behaves differently when typed. It types the stack into the trace window instead of opening a new window.)

14.6 Returning to Top Level

Typing Control-D will immediately take you to the top level from any break window. The functions called before the break will stop, but any side effects of the function that occurred before the break remain. For example, if a function set a global variable before it broke, the variable will still be set afler typing Control-D.

BREAK PACKAGE 14.5
1

----- Next Message -----

Date: 19 Dec 91 15:51 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.155149pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11668>; Thu, 19 Dec 1991 15:51:54 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:51:49 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

27. WINDOWS AND REGIONS

27.1 Windows

Windows have two basic parts: an area Ofi the screen containing a collection of pixels, and a property list. The window properties determine how the window looks, the menus that can be accessed from it, what should happen when the mouse is inside the window and a mouse button is pressed, and soon.

27.1.1 CREATEW

5ome of the window's properties can be specified when a

window is created with the function CREATEW. In particular, it is easy to specify the size and position of the window; its title; and the width of its borders.

(CREATEW region title borderw'idth)

Region is a record, named REGION, with the fields left, botto:. width, and height. A region describes a rectangular area on the screen, the window's dimensions and position. The fields left and bottoø refer to the position of the bottom leff corner of the region on the screen. Vi dth and height refer to the width and height of the region. The usable space inside the window will be smaller than the width and height, because some of the window's region is consumed by the title bar, and some is taken by the borders.

Title is a string that will be placed in the title bar of the window. Bordervvidtfr is the width of the border around the exterior of the window, in number of pixels.
For example, typing:
(SETQ ~.WIN~ CREATEW
(CREAT RE6IS loo 150 300 200)
THIS Is ~ r"w ilIN~ø)

produces a window with a default borderwidth. Note that you did not need to specify all the window's properties. (See Figure 27.1.)

wiNDows AND REGIONS 27

WINDOWS

ø ,.J[1.lGJ'J (øflf,,Tfff i'øCRE"TEPE,IrnN jvjw 5- '9;, ø"~~i
"TriI;' I> My ij'lN 'ff Ibl&U'M' ii

(~[N&JwlM2.65554

FigUre 27.1. Creating a Window

In fact, if (CREATEW) is called without specifying a region, you will be prompted to sweep out a region for the window. (See Section 10.2, Page 10.2.)

27.1.2 WlNDOWPROP

The function to access or add to any property of a window's property list is WIliDOVPROP.
(WIN~PR0P window property <value>)

When you use WIKDOWPROP with only two arguments - window and property - it returns the value of the window's property. When you use wIKOOVPROP with all three arguments - window, property and value - it sets the value the window's property to the value you inserted for the third argument.

For example, consider the window, NY WINDOW, created using (CREATEW). TITLE and REGION are both properties. Type
(ilI*~PW :.uI~ 'TITLE)

and the value of MY.WlNDOW's TITLE property is returned, "THIS 15 MY OWN WINDOW". To change the title, use the WINDOWPROP function, and give it the window, the property title, and the new title of the window.
(wIK~PW ~.uI~ 'TITLE øP FIRST ilIK~ø)

automatically changes the title and automatically updates the window. Now the window looks like Figure 27.2.

27.1 w1N00WS AND REG~NS

WINDOWS

7øt'WINDOWfROP NV WINDOW TITLE)
IS NV OWN WINDOW"

s.(WINDOWPROP NY.WlNDOl4 TITLE 'øQY FIRST WINDOW")
THIS IS M\' OWN WINDOW"
4'.

FigUre 27.2. TITLE is a Window Property

Altering the region of the window, NY. VINDOV, is also be done with vINDOWPROP, in the same way you changed the title. (Note: changing either of the first two numbers of a region changes the position of the window on the screen. Changing either of the last two numbers changes the dimensions of the window itself.)

27.1.3 Getting windows to do things
Four basic window properties will be discussed here. They are CURSORINFN, CURSOROUTFN, CURSORffOVEDFN, and BUTTONEVENTFN.

A function can be stored as the value of the CURSORlNFN property of a window. It is called when the mouse cursor is moved into that window.

Look at the following example:

(1) First, create a window called MY.WlNDOW. Type:
(SETQ P.WINDQW
(CREATEI

(cREATERE6Ia 200 200 200 200)
"THIS WIllDOW WILL IREMlø))

This creates a window.

(2) Now define the function SCREAMER. It will be stored on the property CURSOR1NFN. (Notice that this function has one argument, WlNDOWNAME. All functions called from the property CURSOR1NFN are passed the window it was called from. So the value of MY. WINDOW is bound to WlNDOWNAME. When it is called, SCREAMER simply rings bells.
(DEFIN—Q (ScREMER (WIK~~E)
RIilBELLS)

PROlPTPRIlT TAT - IT WDRFSI")

RIKBELLS)))

(3) Now, alter that window's CURSORINFN property, so that the system calls the function SCREAMER at the appropriate time. Type:

WlNDow5 AND REGIONS 273

WINDOWS

(WIN~PRoP P.wINI;0II 'cuR~RIafø
(F~IIk:TIK IR~R))

(4) Affer this, when you move the mouse cursor into MY.WlNDOW, the CURSORINFK property's function is called, and it rings beJls tvvice.

CURSORINFN is one of the many window properties that come with each window - just as REGION and TITLE did. Other properties include:

CURSOROUTFN The function that is the value of this property is executed when the cursor is moved out of a window;

CURSORMOVEDFN the function that is the value of this property is executed when the cursor is moved while it is inside the window;

BUTTONEVENTFN the function that is the value of this property is executed when either the Ieff or middle mouse buttons are pressed (or released).

Figure 27.3 shows MY.WlNDOW's properties. Notice that the CURSORINFK has the function SCREAMER stored in it. The properties were shown in this window using the function INSPECT. INSPECT is covered in Chapter 32.
. . ' 1 ø
GREEN NIL

HI NOo'rtENTR'[FN O liE. TT'(PRObES
PRflIESS NIL
','181)ROER 4
NEWREL'I)NF4 NIL

'NTITLE øTHIS 'ffiINDOW 'tILL .QCREAn!"
MOlEfN NIL
CLOSEFN NIL
HORIZOCROLL'.yIND1)',t NIL
"ER1L'ROLLNINoO'ff NIL
c.u'ROLLFN NIL
H)RI=J-'cRlLLREG NIL
":'ERTSCR)LLREU NIL
USERDATA NIL
E!'!TENT NIL
REOH4PEFN NIL
REPAINTFN NIL
L'URSORttOvEDFN NIL
CURSOROUTFN NIL
CURSORINFN SCCE'øThER
RIGHTBUTTONFN NIL
BU1FONEVENTFN TOTOPU
REG 12J0 "L)9 øJ~ '36!
SavE (BITMAP~ø!3,1jo52!
NE~('t (WIflD1)'-'1j55,1'lj'..ø8
DSP ~5TRE>M\,ø~øF,jjjj~4

Figurø 27.3. Inspeaing MY.wlNDow for MouseRelated Window Properties
You can define functions for the values of the properties CURSOROUTFK and CURSORMOVEDfN in much the same way as you did for CURSORINfN. The function that is the value of the property BUTTOHEVENTFN, however, cab be specialized to respond in different ways, depending on which mouse button is pressed. This is explained in the next section.

27.1.3.1 BUtrON—VENTFN

BUTTONEVENTFK is anothqr property of a window. Tho function that is stored as tflø valuø of this property is called when tho mouso is insidø tho window, and a mouso button is pressed. As an examplø of how to usø iL type:

27A ~N00wS ANO REGIONS

witurows

(wI~PKP :.ilIK~ 'euTTW"EKTtr
(F~TI5 ScREAøER))

When the mouse cursor is moved into the window, bells will ring because of the CURS0RlNFN, but it will also ring bells when either the Jeff or middle mouse button is pressed. Notice that the right mouse button functions .5 it usually does, with the window manipulation menu. If only the left button should evoke the function SCREAMER, then the function can be written to do just this, using the function MOUSESTATE, and a form that only NOUSESTATE understands, ONLY. For example:
(DEFIKEQ

(SCRE|ERZ WIK~)
(if ESTATE (aLY LEFT))
thøa (RIKB—LLS))))

In addition to (ONLY LEFT), MOUSESTATE can also be passed (ONLY MIDDLE), (ONLY RIGHT) or combinations of these (e.g. (OR (ONLY LEFT) (ONLY MIDDLE))). You do not need to use ONLY with MOUSESTATE for every application. ONLY means that that button is pressed and no other.
If you do write a function using (ONLY RIGHT), be sure that your function also checks position of the mouse cursor. Even if you want your function to be executed when the mouse cursor is inside the window and the right button is pressed, there is a convention that the function DOVINDOWCOM should be executed when the mouse cursor is in the title bar or the border of the window and the right mouse button is pressed. Please program your windows using this tradition! For more information, please see the Intertisp-D Reference Manual, Volume 3, Chapter 28, Pages 7 and 28.

Please refer to the Intertisp Reference Manual, Volume 3, Chapter 28, for more detail and other important functions.

27.1.4 Looking at a window's prOperties
INSPECT is a function that displays a list of the properties of a window, and their values. Figure 27.3 shows the INSPECT function run with MYøWINDOV. Note the properties introduced in CREATEW: WBORDER is the window's border, REG is the region, and WTITLE is the window's title.

27.2 Regions

A region is a record, with the fields LEFT, BOTTOM, WIDTH, AND HEIGHT. LEFT and BOflOM refer to where the bottom leff hand corner of the region is positioned on the screen. WIDTH and HEIGHT refer to the width and height of the region.
CREATERE6ION creates an instance of a record of type REGION. Type:

(SETO ~.RE6Ia (CREATERESIl 15 loo 200 450))

REGIONS

to create a record of type REGION that denotes a rectangle 200 pixels high, and 450 pixels wide, whose bottom leff corner is at position (15, 100). This record instance can be passed to any function that requires a region as an argument, such as CREATEV, above.

a,. WlN00WS ANO REGIONS

----- Next Message -----

Date: 19 Dec 91 15:59 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.155935pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11672>; Thu, 19 Dec 1991 15:59:45 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:59:35 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

28. WHAT ARE MENUS?

While Interlisp-D provides a number of menus of its own (see Section 7.1, Page 7.2), this section addresses the menus you wish to create. You will learn how to create a menu, display a menu, and define functions that make your menu useful.
Menu's are instances of records (see Chapter 24). There are 27 fields that determine the composition of every menu. Because Interlisp-O provides default values for most of these descriptive fields, you need to familiarize yourself with only a few that we describe in this section.

Two of these fields, the TITLE of your menu, and the ITEMS you wish it to contain, can be typed into the InterlispøD Executive window as shown below:

NIL

33'(.ETO MY. MEN (cRE'"TE ME/lb
TiTLE ,.PLE~~SE CHCio8— ONE OF THE
ITEMS"

ITEMS (0,LlIT NE,T-l)UE;STION
NE;~T-TOPIL SEE-TOPIC;5'JJJ
,rMENU!,#c4, ij'ø:'3jH

Figure 28.1. Creating a menu

Note that creating a menu does not display it. MY.MENU is set to an instance of a menu record that specifies how the menu will look, but the menu is not displayed.

28.1 Displaying Menus

Typing either the MENU or ADDNENU functions will display your menu on the screen. MENU implements pop-up menus, like the Background Menu or the Window Menu. ADDMEHU puts menus

into a semi-permanent window on the screen, and lets you select items from it.

(MENU MENU POSITION) pops-up a menu at a particular position on the screen.
Type:

(*EKU MY.ffI KIL)

to position the menu at the end of the mouse cursor Note that the POSITION argument is NIL. In order to go on, you must either choose an item, or move outside the menu window and

WHAT ARE MENUS' 281

DISPLAYING MENUS

press a mouse button. When you do either, the menu will disappear. If you choose an item, then want to choose another, the menu must be redisplayed.

(ADONENU menu window position) positions a permanent menu on the screen, or ;n an existing window.

Type:

(ADlEKU P.*EI)

to display the menu as shown in Figure 28.2. This menu will remain active, (will stay on the screen) without stopping all the other processes. Because ADONEliU can display a menu without stopping all other processes, it is very popular in users programs. If window is specified, the menu is displayed in that window. If window is not specified, a window the correct size for the menu is created, and the menu is displayed in that window.
If position is not specified, the menu appears at the current position of the mouse cursor.

NE..TQøUESIlCN

3EEToPIC> .
.

Figure 28.2. A Simple Menu, displayed with AooNriU.

28.2 Getting Menus to DO Stuff
One way to make a menu do things is to specify more about the menu items. Instead of items simply being the strings or atoms that will appear in the menu, items can be lists, each list with three elements. (See Figure 28.3.) The first element of each list is what will appear in the menu; the second expression is what is evaluated, and the results of the evaluation returned, when the item is selected; and the third expression is the expression that should be printed in the Prompt window when a mouse button is held down while the mouse is pointing to that menu item. This third item should be thought of as help text for the user. If the third element of the list is NIL, the system responds with "Will select this item when you release the button".

JGJ WHAT AR5 MENUS?

GErn~ MENUS TO DO STUFF

NIL

17+(SETQ Nv.MENU2 (SR—ATE MENU

TITLE "PLEASE LHOOSE ONE OF TfIE ITEMS"
I~,EMS '(VQUIT
(PRINT "STOPPEO" \
"LHOOSE THIS TO 5O~ø",'

(NE\T-QUESTIOH

(PRINT "HERE IS TME NE.'\'T QLl—STIOH .
øu'HOOSE THIS TO ~E lSKED THE NE."T QUESTION"',

iNE!~T-TOPIL

(PRINT øøHERE IS THE NE'~T TOPIL .
"C.HOOSE THIS TO KOv— OH TO THE NE'\T SueJELT"' '1

(SEE-TOPICS

(PRINT "THE FOLLOYIN6 HA'\E NOT e.EEN L—ARNEO"',
*CHOOSE THIS TO SEE THE TOPICS NOT YET LErtRNEO"l 'ii
ø~~MENU,'#5~. '.5~5j
1qL(cl&MENL MY. MEtlU:'
,rNIN&El'~~~4', 175350
14

Firnre 28.3. Creating a menu that will do things, then displaying it with the funttion ADDNEHU

Now when an item is selected from KY.KENU2, something will happen. When a mouse button is held down, the expression typed as the third element in the item's specification will be printed in the Prompt window. (See Figure 28.4.)

NE7.T.'JUE'=TlE'r~J
SEE-TOPIC'

Fiqrnre 28.1. Mouse Button Held Down While Mouse Cursor SeIe~
NEXT-QUESTIoN

When the mouse button is released (i.e. the item is selected) the expression that was typed as the second element of the item's specification will be run. (See Figure 28.5.)

Y-'OUE'Tl"N
'EETOPlr"'
"HERE IS THE NEXT ilUETION.
Figure 28.5. NEXT-QUESTION Selected

WHAT ARE MENUS' 283

GEHlNG MENUS TO DO STUFF

28.2.1 _ The WHENH—LDFN _ and WNENSEL—CTEDFN fields of a _ menu
Another way to get a menu to do things is to define functions, and make them the values of the menu's WHENHELDFN and WHENSELECTEDFN fields. As the value of the WHENHELDFN field of a menu, the function you defined will be executed when you press and hold a mouse button inside the menu. As the value of the WHENS—L—CTEDFN field of a menu, the function you defined will be executed when you choose a menu item. This example has the same functionality as the previous example, where each menu item was entered as a list of three items.

As an example, type in these two functions so that they can be executed when the menu is created and displayed:

```
(DEFIKEQ L—CTED

(SELEcfQPiNTEENNUJSENHENHELO (ITEl.S—LECTED a:. FROM BUTT:. PRESSED)
QUIT (PROMPTPRIKT øcHOOSE THIS TO sToPø))

NEXT-QUESTION (PROMPTPRIKT CHOOSE THIS TO BE ASKED TNE NEXT QUESTION-))
NEXT-TOPIC PROMPTPRINT øCHOOSE THIS TO MOO,E a TO THE NEXT SUBUIECTø))
SEE-TOPICS PROMPTPRINT øCHDOSE THIS TO SEE THE TOPICS NOT YET L—ARNEDø))
ERROR (PROM TPRIKT NO liTCH FOUNDø)))))

(DEFINEQ WENSELECTED (ITEM.SELECTED MENU. FROM 8UTT:.PRESSED)
QUIT (PRINT øSTOPPEDø))

NEXT-QU RINT "HERE IS THE NEXT QUESTION...))
NEXT-T øHERE IS THE NEXT TOPIC. .
- PICS PRINT øTHE FOLLONIK HAVE NOT 8EEN LEARNED. ..ø

ERROR (PRONFTPRINT NO liTCH FOUND)))))
```

Now, to create the menu, type:

```
(SETQ MY.NE:3 (CREATE NE:
TITLE øPLEASE CHOOSE :E OF THE ITEMSø
ITEK '(QUIT NEXT-QUESTION NEXT-TOPIC SEE-TOPICS)
NHENHELDFN (FUNCTIK MY.NENU3.NHENHELD)
fflENSELECTEDFN (FUNCTION NY, .MENU3 .fflENSELECTED)))
```

Type

```
(ADDMENU MY.MENU3)
```

to see your menu work.

NOW, due to executing the WH—NNELDFN function, holding down any mouse button while pointing to a menu item will display an explanation of the item in the prompt window. The screen will once again look like Figure 28.4 when the mouse button is held when the mouse cursor is pointing to the item NEXT-TOPIC.

Now due to executing the WHENSELECTEDFN function, releasing the mouse button to select an item will cause the proper actions for that item to be taken. The screen will once again look like Figure 28.5 when the item NEXT-TOPIC is selected.
The crucial thing to note is that the functions you defined for WHENHELDFN and WHENSELECTEDFN are automatically given the following arguments:

(t) the item that was sølected, ITEM. SELECTED;
(2) the menu it was selected from, MENU. FROM;
(3) and the mousø button that was pressed BUTTON PRESSED.
Hotø: thesø functions, *Y.NENU3.fflENflELO and ffY.KEKUJ.ilHEKSELCTEO, wøre quoted using FUKCTIOK instead of QUOTE both for program røadability and so that the

21.1 ~YAR1".NUs?

GETTlMG MENUS TO 00 STUFF

compiler con produce foster code when the program is compiled. It is good style to quote functions in Intertisp by using the function FUNCTION instead of QUOTE.

28.3 Looking at a menuøs fields
INSPECT is a function that displays a list of the fields of a menu, and their values. The Figure 28.6 shows the various fields of NY .NENU3 when the function (INSPECT NY øNENU) was called. Notice the values that were assigned by the examples, and all the defaults.

\JN"PELT NY liENl./3øøl
øl1IHDU'wJ#'1, 54øscj

NENllPECICNB1:TTi=fl o

Imrni';E (ø!VINDLlrtø¤#b1.l‡5lSjl

ø t1)UlT HEø~T-LløL'E'TI1=1N 'ø'-Ti'iFl'ø ET
ø MENUPOffoø'

ø ANUEAFF'ETFLL: NIL

ffENUEQHT i:FclNTPc:::cf IpTclFt -a
TITLE 'øPLEAL'E CHil.l 'HE ,iF THE ITE
ø ffEHlJoFF6ET A
LECTEDFN fly flEflJ,ø h.ørtEf:EL.FCTE—l

'1flE'flELDFH NV flEPlLl3 \ørtEHHELJP
ø ENl)NHELoFH l:LFF'RCHPT

ø flENOFEEOe4l,'r.FLG NIL
Figure 28.6. The Fields of MY.MENU3

WHAT ARE MENUS' 285

----- Next Message -----

Date: 19 Dec 91 16:10 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.161052pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11680>; Thu, 19 Dec 1991 16:10:56 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:10:52 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

29. lilTMAPS

A bitmap is a retangular array of dots. The dots are called pixels (for picture elements). Each dot, or pixel, is represented by a single bit. When a pixel or bit is turned on (i.e. that bit set to 1), a black dot is inserted into a bitmap. If you have a bitmap of a floppy on your screen, (Figure Figure 29.1), then all of the bits in the area that make up the floppy are turned on, and the surrounding bits are turned off.

FLOPPY
(Ia b~JwP-
~',5,,Bh

(t-:)o

Figure 29.1. Bitmap of a Floppy

BITNAPCREATE creates a bitmap, even though it can't be seen.
(BIfflPCRDTE width height)

If the width and height are not supplied, the system will prompt
you for them.

EDZTBN edits the bitmap. The syntax of the function is:
(EDITl bitmapname)

Try the following to produce the results in Figure 29.4:
l~SoETiQr:~røB!sTHituPbB~I~PcRDTE eo 40))

To draw In the bitmap, move the mouse into the gridded section of the
bitmap editor, and press and hold the leff mouse button. Move
the mouse around to turn on the bits represented by the spaces
in the grid. Notice that each space in the grid represents one
pixel on the bitmap

To erase Move the mouse into the gridded section of the bitmap editor,
and press and hold the center mouse button. Move the mouse
around to turn off the bits represented by the spaces in the
gridded section of the bitmap editor.

To work on a different section Point with the mouse cursor to the picture of the actual bitmap
(the upper left corner of the bitmap editor). Press and hold the

Jeff mouse button. A menu with the singJe item, ttove will
appear. (See Figure 29.2.) Choose this item.

. .

Figure 29.2. Move the mou5e cursor to the Dtcture of the bitmap. Press and hold
the Iek mouse button. and the Move menu will appear

You will be asked to position a ghost window over the bitmap.
This ghost window represents the portion of the bitmap that you
are currently editing. Place it over the section of the bitmap that
you wish to edit. (See Figure 29.3.)

. .
.

. .

. ... .
.... I .

29.3. .. J=.. :. II.lI:.:;;. _ . .
f1ure Affer you choose move. yoU will be asked to position a ghost

window like this one. Position it by clicking the leff mouse button when the

ghost window is over the part of the picture of the bitmap you would like to edit.
To end the session 8ring the mouse cursor into the upper-right portion of the
window (the grey area) and press the center button. Select OK
from the menu to save your artwork.

29) .lY~

r’. alTMAps

.:: 5”iSETQ ffy IllNAP (I[TNAPcPEATE OR GO)’
j:y.IlfM&P osøt\
,A.BlTMAPlø6’,1.q;łlO
58oi,EOIlBM my.IlTNAP\

- -
.

.
-A

fr.j:
’

= ""~
=
. ~.
.

. .

F~ure 29.4. Editing a Bitmap

BITBLT is the primitive function for moving bits (or pixels) from one bitmap to another. It extracts bits from the source bitmap, and combines them in appropriate ways with those of the destination bitmap. The syntax of the function is:

(BITBLT sourcebitmap sourcelefl sourcebottom destinationbitmap destinationleft destinationbottom width height sourcetype operation texture clippIngregion)

Here’s how it’s done - using MY.BlTMAP as the sourcebitmap and MY.WlNDOW as the destinationbitmap.’
(BITBLT rn.BITll4P NIL NIL

P.wIN~ NIL NIL KIL NIL ’INPUT ’REPuCE)

Note that the destination bitmap can be, and usually is, a window. Actually, it is the bitmap of a window, but the system handles that detail for you. Because of the IlLs (meaning "use the default"), MY.BlTMAP will be BlTBLT’d into the lower right hand corner of MY.WlNDOW. (See Figure 29.5.)

BlTMAPS 293

~17MAP5

98’(BITBLT KY Strap NIL NIL my ,1(10p,, FL ’IL NIL HIL Tipil’ P.—PLlfi

(~=l’,

Figure 29.5. 9ITBLTng a Bitmap onto a Window

Here is what each of the SlTBLT arguments to the function mean:

sourcebitmap the bitmap to be moved into the destinationbitmap
sourcelett a number, starting at O for the Jeff edge of the sourcebitmap, that tells SITBLT where to start moving pixels from the sourcebitmap. For example, if the leftmost 10 pixeis of

sourcebitmap were not to be moved, sourceleft should de 10
The default value is O.

sourcebottom a number, starting at O for the bottom edge of the
sourcebitmap, that tells BIT6LT where to start moving p1'xels
from the sourcebitmap. For example, if the bottom 10 rows of
pixels of sourcebitmap were not to be moved, sourcebottom
should be 10 The default value is O.

destinationbitmap the bitmap that will receive the sourcebitmap. This is offen a
window (actually the bitmap of a window, but Interlisp-b takes
care of that for you).

destinationleff a number, starting at O for the leff edge of the
destinationbitmap, that tells BITBLT where to start placing
pixels from the sourcebitmap. For example, to place the
sourcebitmap 10 pixels in from the Jeff, destinationleft should be
10. The default value is 0.

destinationbottom a number, starting at 0 for the bottom edge of the
destinationbitmap, that tells BITBLT where to start placing
pixels from the sourcebitmap. For example, to place the
sourcebitmap 10 pixels up from the bottom, destinationbottom
should be 10. The default value is 0.

width how many pixels in each row of sourcebitmap should be moved.
The samc amount of space is used in destinationbitmap to
receive the sourcebitmap. If this argument is NIL, it defaults to
the number of pixels from sourceleft to the end of the row of
sourcebitmap.

height how many rows of pixels of sourcebitmap should be moved. The
same amount of space is used in destinationbitmap to receive
thq sourtebitmap. If this argument is NIL, it defaults to the
number of row; from sourcebottom to tho top of the
sourcebitmap.

sourcetypø rofors to onø of throø ways to cofivørt thø sourcebitmap for
writing. For now, just usø 'INPUT.

29.ø o~ps

, , , , , , , , , , , , , , „ , , "' , , , , , , , , , , , , , ,

OIlMAPS

operation refers to how the sourtebitmap gets BlTBLT'd on to the
destinationbitmap. 'REPLACE will BLT the exact sourcebitmap.
Other operations allow you to AND, OR or XOR the bits from the
sourcebitmap onto the bits on the destinationbitmap.
texture Just use NIL for now.
clippingregion just use NIL for now.

Por more information on these operations, see the Interlisp-D
Reference Manual, Volume 3, Chapter 27, Page 14.

Sourcebitmap, sourceleft, sourcebottom, destinationbitmap,
destinationleft, destinationbottom, width and height are shown
in Figure 29.6.

Destination Bitmap
Source Bitmap
FLOPPY
tlcblffkUP'

3/S/Bh height

e./,o

width

Source leh. Source bottom. The "x y coordinates in terms of the source (OOforthewhoiesource).

Destination Jeff, Dertination Bottom. The ,,x y" coordinates in terms of the destination bitmap. (00 to put the source bitmap in the Ieft bottom corner of the dertination bitmap).

Figure 29.6. BITBLT'ed Bitmap of a Floppy

BITMAPS 295

----- Next Message -----

Date: 19 Dec 91 16:16 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.161653pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11679>; Thu, 19 Dec 1991 16:16:57 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:16:53 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

30. DlSPLAYSTREAMS

A displaystream is a generaJized "place to display". They determine exactly what is displayed where. One example of a displaystream is a window. Windows are the only displaystreams that will be used in this chapter. If you want to draw on a bitmap that is not a window, other than with BITBLT, or want to use other types of displaystreams, please refer to the Interlisp-D Reference Manual, Volume 3, Chapter 27.

This chapter explains functions for drawing on displaystreams: DRAWLINE, DRAWTO, DRAVCIRCLE., and FILLCIRCLE. In addition, functions for locating and changIng your curreAt position in the displaystream are covered: DSPXPOSITIOH, DSPYPOSITION, and NOVETO.

30.t Drawing on a Displaystream
Examples will show you how the functions for drawing on a display stream work. First, create a window. Windows are displaystreams, and the one you create will be used for the examples in this chapter. Type:
(SETO EwPLE.wIN~ (CREATEI))

30.1.1 DRAWLlNE

DRAWL IRE draws a line in a displaystream. For example, type:
(DliVLIKE 10 IS loo 150 S øIlERT ExMPLEwIN~)
The results should look like this:

Figure 30.1. The line drawn onto the displayrtream, ExAMPLEwlNDoW

DlSPLAYSTREAMS 30

DRAWING ON A DlSPLAYsTaE:M
The syntax of DRAWL1NE is

(Dli~IKE xl yl x2 y2 width opera tion stream ø)
The coordinates of the Jeff bottom corner of the displaystream
areOO.

xl and yl are the x and y coordinates of the beginning of the line;
x2andy2 are the ending coordinates of the line;
width isthe width of the line, in pixels

operation is the way the line is to be drawn. INVERT causes the line to
invert the bits that are already in the displaystream. Drawing a
line the second time using INVERT erases the line. For other
operations, see the Interlis~D Reference Manual, Volume 111,
Page 27.15.

stream is the displaystream. In this case, you used a window.

30.1.2 ORA~O

DRAWTO draws a line that begins at your current position in the
displaystream. For example, type:
(Dli~O 120 135 5 'IrvERT E~LE.*IH~)
The results should look like this:

Figuro 30.2. Another line drawn onto the displaystream, ExAMPLEøWlNDowø
The syntax of ORAWTO is

(oliilT0 x y width operation stream i)

The line begins at the current position in the displaystream.
x is the x coordinate of the end of the line;
y is they coordinate of the end of the line;
width is the width of the line

operation is the way the lino is to be drawn. INVERT causes the line to
invert the bits that aro already in tho displaystream. Drawing a
line the second time using INVERT erases the line. For other
operations, see the lnteHi~O Reference Manual, Volume Ill,
Page 27.15.

stream is the displaystreom. In this case. you used a window.

30.2 llPLAYSTQCANT

DRAW1NG ON A D15PLAr5~E~
30.1.3 DRAWClRCLE

DRAWCIRCLE draws a circle on a displaystream. To use it, type:
(0li~I~LE 150 100 so '(~RTICAL 5) KIL E~LE .VI~)
Now your window, EXAMPLE.WlNDOW, should look like this:

Flurø 30.3. The circle drawn onto the displaystream. EXAMPLE WINDOW
The syntax of DRAWCIRCLE is

(0li~IEL— centerx centery radius brush dashing stream)
centerx is the x coordinate of the center of the circle
centery is they coordinate of the center of the circle
radius is the radius of the circle in pixels

brush is a list.- The first- item of the list is the shape of the brush. Some of your options include ROUND, SQUARE, and VERTICAL. The second item of that list is the width ofthe brush in pixels.
dashing is a list of positive integers. The brush is "on" for the number of units indicated by the first element of the list, "off" for the number of units indicated by the second element of the list. The third element specifies how long it will be on again, and so forth. The sequence is repeated until the circle has been drawn.
stream is the displaystream. In this case, you used a window.

30.1.3.1 FlLLClRCLE

FILLCIRCLE draws a filled circle on a displaystream. To use it, type:

(FILLCIRCLE 200 150 10 6liY~DE ExlPLE.wIli~)
EXAMPLE.WlNDOW now looks like this:

DlSPLAYSTREAMS 303
l

DRAWING ON A DISPLAYSTREAM

Figure JO.t A filled circle drawn onto the displaystream, EXAMPLE WINDOW
The syntax of FILLCIRCLE i5

(FILLCIRCL— centerx centery radius texture stream)
centerx is the x coordinate of the center of the circle
centery is theycoordinate of the center of the ci rcle
radius is the radius of the circle in pixels

texture is the shade that will be used to fill in the circle. Interlisp-D provides you with three shades, WHlTESHADE, BLACKSHADE, and GRAYSHADE. You can also create your own shades. For more information on how to do this, see the Interlisp-D Reference Manual, Volumelll, Page 27.7.

stream is the displaystream. In this case, you used a window.
There are many other functions for drawing on a displaystream. Please refer to the Intertisp-D Reference Manual, Volume 111, Chapter 27.

Text can also be placed into displaystreams. To do this, use printing functions such as PRIffl and PRIN2, but supply the name of the displaystream as the "file" to print to. To place the ten in the proper position in the displaystream, see 5ection 30.2, Page 30.4.

30.2 _ Locating _ and _ Changing _ Your _ Position _ in _ a _ Displaystream
There are functions provided to locate, and to change your current position in a displayitream. This can help you place text, and other images where you want them in a displaystream. This primer will only discuss three of these. There are others, and they can be found in the lnterlis~D Reference Manual, Volume Ill, Chapter 27.

30.4 0lSPLAY¤TREA~

r.

LOCATING AND CHANGING YOUR POSITION IN A DISPLAYSTREAM

30.2.1 DSPXPOSlTlON

DSPXPOSITION is a functiOn that will either change the current x pOsition in a displaystream, or simply report it. To have the function report the current x position in EXAMPLE.WlNDOW, type:

(OSP*PoSlTIoN NIL EXlPLE .ilINDON)

DSPXPOSITION expects two arguments. The first is the new x position. If this argument is NIL, the current position is not changed, merely reported. The second argument is the displaystream.

30.2.2 DSPYPOSlTlON

DSPYPOSITION is an analogous function, but It changes or reports the current y position in a displaystream. As with DSPXPOSlTlON, If the first argument Is a number, the current y position will be changed to that position. If it is NIL, the current position is simply reported. To have the function report the current y position in EXAMPLE.WlNDOW, type:

(DSPYROSITIoN NIL ExlPLE.WIK-~)

30.2.3 MOVETO

The function NOVETO always changes your position in the displaystream. It expects three arguments:
(~-ET0 xystream)

x is the new x position in the display stream
y is the new y position in the display stream

stream is the display stream. The examples so far have used a window.

DISPLAYSTREAMS 30 5

----- Next Message -----

Date: 19 Dec 91 16:30 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.163054pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11682>; Thu, 19 Dec 1991 16:30:58 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:30:54 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

31. FONTS

This chapter explains fonts and fontdescriptors, what they are and how to use them, so that you can use functions requiring fontdescriptors

You have already been exposed to many fonts in Interlisp-D. For example, when you use the structure editor, DEdit, (See Section 11.3.), you noticed that the comments were printed in a smaller font than the code, and :hat CLlSP words (See Section 13.1, Page 13.1.) were printed in a darker font than the other words in the

function. These are only -me of the fonts that are available in Interlisp-D.

In addition to the fonts that appear on your screen, Interlisp-D uses fonts for printers that are different than the ones used for the screen. The fonts used to print to the screen are called DlSPLAYFONTS. The fonts used for prining are called INTERPRESSFONTS, or PRESSFONTS, depending on the type of printer.

31.1 What makes up a FONT?

Fonts are described by family, weight, slope, width, and size. This section discusses each of these, and describes how they affect the font you see on the screen.

Family is one way that fonts can differ. Here are some examples of how "family" affects the look of a font:

CLASSIC This family makes the word "Able" look like this: Able
MODERN This family makes the word "Able" look like this: Able
TERMINAL This family makes the word "Able" look like this: Able
Weight also determines the look of a font. Once again, "Able" will be used as an example, this time only with the Classic family. A font's weight can be:
BOLD and look like this: Able
MEDIUM or REGULAR and look like this: Able
The slope of a font is italic or regular. Using the Classic family font again, in a regular weight, the slope affects the font like this:

ITALIC looks like this: A file
REGULAR looks like this: Able

FONT5 311
1

WHAT MAKES UP A FONT?

The width of a font is called its "expansion". It can be COMPRESSED, REGULAR, or EXPANDED.

Together, the weight, slope, and expansion of a font specifies the font's "face". Specifically, the face of a font is a three element list:

(weight slope expansion)

To make it easier to type, when a function requires a font face as an argument, it can be abbreviated with a three character atom. The first specifies the weight, the second the slope, and the third character the expansion. For example, some common font faces are abbreviated:

MRR This is the usual face, MEDIUM, REGULAR, REGULAR;
MlR makes an italic font. It stands for: MEDIUM, ITALIC, REGULAR;
BRR makes a bold font. The abbreviation means: BOLD, REGULAR, REGULAR;

BIR means that the font should be both bold and italic. BIR stands for BOLD, ITALIC, REGULAR.

The above examples are used so oflen, that there are also more mnemonic abbreviations for them. They can also be used to

specify a font face for a function that requires a face as an argument. They are:

STANDARD This is the usual face: MEDIUM, REGULAR, REGULAR. It was abbreviated above, MRR;

ITALIC This was abbreviated above as MR, and specifies an italic font; BOLD of course, makes a bold font. It was abbreviated above, BRR; BOLDlTALIC means that the font should be both bold and italic: BOLD, ITALIC, REGULAR. It was abbreviated above, BlR.
A font also has a size. It is a positive integer that specifies the height of the font in printers points. A point is, on an 1108 screen, about 1/72 of an inch. On the screen of an 1186, a point is 1/80 of an inch. The size of the font used in this chapter is 10. For comparison, here is an example of a TERMINAL, MRR, size 12 font: Able.

31.2 Fontdescriptors, and FONTCREATE
For InterlispøD to use a fort, it must have a fontdescriptor. A fontdescriptor is a data type in InterlispøD that that holds all the information needed in order to use a particular font. When you print out a fontdescriptor, it looks like this:
[fKTDEIRIPToRjøiø,øs~ø0

Fontdescriptors are created by the function F0NTCREATE. For example,

(F~TCREATE 'flEL~1lCA 12 '~o)

J:

31.2 FOflff

FONTDESCRlPTORS, AND F0NTCREAlE

creates G fontdescriptor that, when used by other functions, prints in HELVETIEA BOLD size 12. Interlisp-D functions that work with fonts Gxpect a fontdescriptor produced with the FONTCREATE function.
The syntax of FONTCREATE is:
(F0KTCREATE family size face)

Remember from the previous section, face is either a three element list, (weight slope expansion), a three character atom abbreviation, e.g. MRR, or one of the mnemonic abbreviations, e.g. STANDARD.

If FONTCREATE is asked to create a fontdescriptor that aJready exists, the existing fontdescriptor is simply returned.

31.3 Display Fonts - Their files, and how to find them
Display fonts require files that contain the bitmaps used to print each character on the screen. All of these files have the extension .DlSPLAYFONT. The file name itself describes the font style and size that uses its bitmaps. For example:
~ERK12.DISPUYFRT

contains bitmaps for the font family MODERN in size 12 points. Initially, these files are on floppies. The files that are used most offen should be copied onto a directory of your hard disk or fileserver. Usually, this directory is called FONTS.
Wherever you put your .DISPLAYFONT files, you should make this one of the values of the variable DISPLAYFONTDIRECTORIES. Its value is a list of directories to search for the bitmap files for

display fonts. Usually, it contains the "FONT" directory where you copied the bitmap files, the device (FLOPPY), and the current connected directory The current connected directory is specified by the atom NIL. Here is an example value of DISPLAYFONTDIRECTORIES:

. - 11
NIL

r~':PI:='pL"'yFnNTDIP,ECTBP,IES

i;!Iøo= ' . =PFIL -FnNT~." (D.~fr):!.LIT'.PFIL
fFLnPF"')- NIL!i
9!ø

Figure 31.1. A valueøfor the atom DISFLAYFONTDIRECTORIES When looking for a .DiSPl.AYFONl file. 'he system will check the F0NT directory on the hard disk. then the top level directory on the hard disk, then the floppy. then the current connected dir8rtory

FONTS 313

INTERPRESS FONT5 - THEIR FILES, AND HOW TO FIND THEM

31.4 _ Interpress _ Fonts _ - _ Their files, _ and _ how _ to _ find _ them
Interpress i5 the format that is used by Xerox laser printers. These printers normally have a resolution that is much higher than that of the screen: 300 points per inch.

In order to format f,Ies appropriately for Output on such a printer, Interlisp must know the actual size for each character that is to be printed. This is done through the use of width files that contain font width information for fonts in Interpress format. Initially, these files (with extension .WD) are on floppies. The files should be copied onto a directory of your hard disk or fileserver.

For Interpress fonts, you should make the location of these files one of the 'values of the variable INTERPRESSFOliToIRFcTORIES. Its value is a list of directories to search for the font viidths files for Interpress fonts. Here is an example value of INTERPRE5SFONTD1RECT0R1ES:
. 11
1'lIL

i?IbdTEFPfiETø=:FnN7PIP:EcTnRI—~,~
.i=~~.~
j:~,~

Figure 31.2. A value for the atom INTERPREssFoNTDIREcToRIEs When looking for a font widths file for an Interpress font, Interlisp-D will cne~ the hard disk.

31.5 Functions for Using Fonts

31.5.1 F0NTPR0P Looking at Font Properties
It is possible to see the properties of a fontdescriptor. This s done with the function FONTPROP. For the following examples, the fontdescriptor used will be the one returned by the function (DEFAULTFONT 'DISPLAY). In other words, the fontdescriptor examined will be the default display font for the system.

There are many properties of a font that might be useful for you.

Some of these are:

FAffILY To see the family of a font descriptor, type:
(FKTPliP (DEFAllLTFoIT 'DISPLAY) 'f~ILY)

SIZE As above, this is a positive integer that determines the height of the font in printer's points. As an example, the SIZE of the current default font is:

31ø ~n

FUNCTIONS FOR USING FONTS

. 11
NIL

Gi,ø(FnNTPROP (DEF~ULTFONT PI~~PLAY)
'.,ø,:'IZE\
is,

Figure 31.3. The value of (he font property SIZE of the default font
ASCENT The value of this property is a positive integer, the maximum height of any character in the specified font from the baseline (bottom). The top of the tallest character in the font, then, will be at (BASELINE # ASCE[VT - l). For example, the ASCENT of the default font is:

ø 1 11
NIL

Aø 4' ,. I!øFnNTPROP if OfF"' ULTFnNT øPI~,~PL~","!'
'~e-rENT:!
q.-

A,5~:

Figure 31.& The value of the font property ASCENT of the default font
DESCENT The DESCENT is an integer that specifies the maximum number of points that a character in the font descends below the baseline (e.g. letters such as "p" and "g" have tails that descend below the baseline.). The bottom of the lowest character in the font will be at (BASELINE - DESCENT). To see the DESCENT of the default font, type:

(FOkTPROP (DEfAULTFKT 'DISPUY) 'DESr:—KT)
HEIGHT HE IGHT is equal to t'DESCENT-ASCENT).
FACE The value of this property is a list of the form, (weight slope expansion). These are the weight, slope, and expansion described above. You can see each one separately, also. Use the property that you are interested in, VEIGHT, SLOPE, or EXPANSION, instead of FACE as the second argument to FONTPROP.

For other font properties, see the Interlisp-D Reference Manual, VolumeIll, Pages 27.27 - 27.28.

31.5.2 5TRlNGWlDTH

It is offen useful to see how much space is required to print an expression in a particular font. The function STRINGVIDTH does this. For example, type:

(STRIKWIDTH "NV thera!ø ('L'NTcREAT— 'UCli 10 'STAKDARD))
The number returned IS how many leff to right pixels would be

needed if the string were printed in this font. (Note that this

F0NTS 31 S

FUNCTIONS FOR USING FONTS

doesn't ju5t work for pixels on the screen, but for all kinds of streams. For more information about streams, see Chapter 30.) Compare the number returned from the example call with the number returned when you change GACHA to TlMESROMAN.

31.5.3 DSPFONT - Changing the Font in One Window
The function DSFF0NT changes the font in a single window. As an example of its use, first create a window to write in. Type:
(SETQ ~.FoNT.WINnaN (CttEATE*))

in the Interlisp-D Executive window. Sweep out the window. To print something in the defau!t font, type:
(PRINT 'HELLO N'f'.FO*T.wIN~)

in the Interlisp-D Executive window. Your window, MY. FONT.WlNDOW, will lOOk sOmething like this:

HELL

Figure 31.5. HELLO, printed with the default font in MY.FONT.WINOOW
Now change the font in the window. Type:
(DSPF0NT (FONTCREATE 'HELVETICA 12 'SOLD) *T.FONT.WINDaN)
in the Interlisp-D Executive window. The arguments to FONTCREATE can be chang~-'d to create any desired font. Now retype the PRINT statement, and your window will look somethinglikethis:

- .
HIL

.q.'~;, PSPFnNT (FnNTrRE~TE 'HEL";'ET1L~
1:'ø øBnLPt

M'tø.FnNT.vINPnWj

l:FnNTPE~1'RIpTnfl~#?.~,. 1-'ø 14 "4
3~~iPR[NT 'HELLO MY.fnflr.l]INoniff)
HELLO

Flgurø 31.L The font iiiMY FONT WINDow, changed
Notice the font has been changedl

J.

31.6 FONtt

FUNfll0NS FOR USING F0Nff
31.5.4 _ Globally Changing _ Fonts _____
There is a library package to globally change the fonts in all the windows. To use it, first load BlG.DCOM. (See Section 8.6, Page 8.4 for how to load a file.)

To change fonts in 311 windows using the package BlG.DCOM, type

(KE*Fo*T <ke~o~>~

There are four keywords for size of fonts to specify. They are

HUGE, BIG, STANDARD, and MEDIUM. For example:
(*E*FKT 'BIG)

sets the fonts in ALL the windows to be a larger size. Note: this package changes the fonts everywhere, including the editor window and system merius It is particularly useful to change the size of the font for demos.

31.5.5 Personalizing Your Font Profile
Interlisp-D keeps a list of default font specifications. This list is used to set the font in all windows where the font is not specifically set by the user (Section 31.5.3). The value of the atom FONTPROFILE is this list. (See Figure 31.7.)

A FONTPROFILE is a list of font descriptions that certain system functions access when printing output. It contains specifications for big fonts (used when pretty printing a function to type the function name), small fonts (used for printing comments in the editor), and various other fonts.

F0NTS 317
I

FUNCTIONS FOR USING FONTS

- . . .
43-FJtlTPRUF[LE

l!' P—F"ULTFClFlT i ,.' ','cH4 LLT;
;ø','~LHk aj
t'TEPMINk'L Sij

'.'BlLPF'lNT :' (H—LlETIl='n' Jo E,PP.;1
'lH—L':'—TIC" L=' BPP,i
'llJPEPtl in' ~FF)

','LITTL—FC'NT 3 ;ttEL'ø?ErIC" ,3,'
iHE - c 1,1p;

i.'BIC-FCNT ~ llnof - hlIP"i
'HE 1.=' BpP.i
'.'HEL'ø'ET .- it' epp:'
'.'IrtoPEPrl -
(J\EPFONT 6oLOFElNT
(C.lMllENTFANT LITTL—Fi)r'T
'.'L"M0P~"FL1flT 61 eFol~! T
i.'.='r'3TEMFeNT',i

'.'CLI~T'~PFUNT BC'LOF')1'lf
i.' CH,,N'3'EF')HT

i,'PPETT\'1?.Cit1F(ji~T Bl.lLC'FllltlT
i.'FCPlTL DEf"ULTFEitiT"
,'Fel'JT"ø 6cLDFcllT;
t.'FCt'1T3 LITTLEFciføIT;
'.f1tlTJ BlCF(.'r'lT',i

i.'FElNT~ S ','HEL',"ETl;,,' 10 81P'.
'lHEL'y'ETlc,, '3 61A)
CfillDEPN a 81P:;

t.'FilNTB 6 'HEL";'ET16~ 10 8RP',i
'.'HEL'."ET1C~ L'~ BAA"'
llPEPN 3 BAA]

Fi)NT7 ? c'"i'.H~ 1:ø"
:e-"Ln~ 1:ø!'

'.TERMItl,'L 1;".,,!,
5a,

Figure 31.7. The value of the atom FONTPROFlLE

The list is in the form of an a5sociation list. The font class names, (e.g. DEFAULTFONT, Or SOLDFONT) are the keywords of the association list. When a number follows the keyword, it is the font number for that font class.

The lists following the font class name or number are the font specifications, in a form that the function FONTCREATE can use. The first font specification list affer a keyword is the specification for printing to windows. The list, (GACHA 10), in the figure above is an example of the default specification for the printing to windows. The last two font specification lists are for Press and Interpress file printing, respectively. For more information, see the lnterlis~D Reference Manual, Volume 3, Chapter 27.

Now, to change your default font settings, change the value of the variable FONTPROFIL—. lnterlis~D has a list of profiles stored as the value of the atom FONTDEFS. Choose the profile to use, then install it as the default FONTPROFILE.
Evaluate the atom FONTDEFS and notice that each profile list begins with a keyword. (5ee Figure 31.8.) This keyword corresponds to the size of the fonts included. BIG, SMALL, and STANDARD are some of :he keywords foT profiles on this list - SMALL and STANDARD appear in Figure 31.8.

31.8 F0Htt
1

FUNCTl0NS F0R USING F0NTS
[[SMALL cFONTPRQFlLE
(DEFALlLTFONT l (TERMINAL
8)

tøUaCHA 8)
"TERmIHAL 8))

(8OLPFL~NT (Mi!OERtt 3 BRR)
\HELY'FTIL" 6 BRR)
ltl\flEfiH 8 BRfi))
1 LITTLEFCNT ~'ø
(hllCiERN 8 MIR)
lHEL'v'ETIu"" 8 MIR)
iMCiPERN ,q, MIR))
(TIN\FONT a IhllOERN a)
to,'F..H" ~)
hll!nEr.H 6

iBIrFnNT j (;',nPF~N 1P BFR)
'!HE"'.'LETIcA lG BRF)
hlrPEF;11 16 ~RP)

iTE.\TFrNT r '.,6LM"~'.lC 13)
'iTIhlE:'Pnn,,"N In)
i.LL~.~:IC lot)
!'TE\TBnLPFnNT
tCL~~CIC 16 Bfifi.,'
~TIME.';RL1MAN

1P BfiR)

tP:LAc.~,Ir 16 BRR]
[cT~NPARP (FDNTPrnPiLE
(PEF"ULTFnNT 1

Figure 31.8. Part of the value of the atom FONTDEFS
To install a new profile from this list, follow the following example, but insert any keyword for BIG.

To use the profile with the keyword BIG instead of the standard one, evaluate the following expressioh
(FOMTSET 'BIG))

Now the fonts are permanently replaced. (That is, until another profile is installed.)

FONIS 319
1

r.

FUNCTIoNS FOR USING F0NTS

[[SMALL cFONTPROFlLE
(OEFALlLTPONT i (TERMINAL
6)

*U'acHA 6)
tøTERmIHAL 6))

(SOLPFL~NT (M'1.OERN 6 BRR)
tHELY'FTIL"' 6 BRR)
Ihll!OER'H qL BRR))
i LITTLEFCNT ~"
(MlcERN 6 MIR)
lHEL'v'ETIu"" 6 MIR)
iMCiOERN ,qø MIR))
(TIN\FONT a IhllOERN a)
U,,F.,H" aj
hll!nEr.N 6

iBIrFnNT J ';;1nPF~N 1P BFR)
'!HE"."LETICA 16 BRF)
hlrPEF;i1 16 ~fiP) !'

i TE.\TFrNT r 6L"~.'lc 1'~)
liTIhlE;:pnMN In)
i.LL~.>:Ic In:)
!'TE\TBnLPFnNT
t CLA~C 1 16 Bfifi
jTIME.';ROMAN
1P BfiR)

\P:LAc.~.Ir 16 BRR]
[<~T~NPARP (FlNTPRnPILE
(PEF"ULTFnNT 1

Figure 31.8. Part of the value of the atom FONTDEFS
To install a new profile from this list, follow the following example, but insert any keyword for BIG.

To use the profile with the keyword BIG instead of the standard one, evaluate the following expressioh

(FlTSET 'BIG))

Now the fonts are permanently replaced. (That is, until another profile is installed.)

FONTS 319
1

----- Next Message -----

Date: 19 Dec 91 16:35 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.163540pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11681>; Thu, 19 Dec 1991 16:35:49 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:35:40 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

12. YOUR IN\T FILE

Interlisp-D has a number of global variables that control the environment of your 1108 or 1186. Global variables make it easy to customize the environment to fit your needs. One way to do this is to develop an "IN\Tø' file. This is a file that is loaded when you log on to your machine. You can use it to set variables, load files, define functions, and any other things that you want to do to make the Interlisp-D environment suit you.

Your lnit fi'e could be callecl IN\T, IN\T.L\SP, IN\T.USER, or whatever the convention is at your site. There is no default name preferred by the system, it just looks for the files listed in the variable USERGREETFILES, (see below). Check to see what the preference is at your site. Put this file in your directory. Your directory name should be the same as your login name.
The IN\T file is loaded by the function GREET. GREET is normally run when Interlisp-D is started. If this is not the case at your site, or you want to use the machine and Interlisp-D has already been started, you can run the function GREET yourself. If your user name was, for example, TUR\NG, then you would type:
(GREET 'TURIK)

This does a number of things, including undoing any previous greeting operation, loading the site init file, and loading your init file. Where GREET looks for your IN\T file depends on the value of the variable USERGREETFiLES. The value of this variable is set when the system's SYSOUT file is made, so check its value at your site! For example, its value could be:
- . - 11
NIL

3'USERGREETF\LE5

iiiFD5hl,(.LI5PFILES~ USER ;INIT.LISPJ
t1rD5h','.LI5PFILE.>.~INIT.LI5PJ
t',rFLoPPY',INIT.L15—J

i,rosh','øLI5PFILES\ USER .'INIT.U5ERJ
((O.h L FILE.' .IN\T.U.'ER'øj

i(D. . FIL SER INIT:,
i(FLUPP';'j I

F~ure12.1. ApcsstblevalueofUSERGREETFILES.
In each place you see, "> USER >", the argument passed to GREET is substituted i:ito the path. This is your login name if you are just starting Interlisp-D. For example, the first value in the list would have the system check to see whether there was a file, [DSX]<L'SPFlLES>TURlNG>lNlT.LlSP. No error is generated if you do not hcve an INlT file, and none of the files in USERGREETFZLE¤ are found.

Y0UR NIT FILE 12 1

MAKING AN INlT FILE

12.1 Making an lnit File

As described in Section 11.5, Page 11.7, each lnterlis~D program file has a global variable associated with it, whose name is formed by appending "COMS" to the end of the root filename. For any of the standard INlT file names, the variable INlTCOMS is used. To set up an init file, begin by editing this variable. First, type:

(SETQ I*ITco*s '((VAnS)))
Now, to edit the variable, type:
(l z:sicn*s>

A DEdit window wiil appear. This DEdit window is the same as the one called with the function OF, and described in Section 11.3, Page 11.4. This chapter will assume that you know how to use the structure editor, DEdit.

The CONS variable is a list of lists. The first atom in each internal list specifies for the file package what types of items are in the list, and what it is to do with them. This section will deal with three types of lists: VARS, FILES, and P. Please read about others in the lnterlis~D Reference Manual, Volume ll, Chapter 17.
The list that begins with "VAR5'ø allows you to set the values of variables. For example, one global variable is called DEditLinger. Its default value is T, and means that the D—dit window won't close affer you exit DEdit. If it is set to NIL, then the DEdit window will be closed when you exit DEdit. To set it to NIL in your INlT file, edit the VARS.list so that it looks like this:
ø . . .1 1 ø1
((',4R.¤' iOEdirLinger NlLii Her
B~,are
G~lete
Replace
'3yvitch
( )

(out
Undo
Find
5'rtap
Rcprint
Edit

EditCam
Break
Eva

Exit

Figurø 12J. Setting the variable DEdI tLi nge r in INITCONS. Notice that inside the vars list, there is yet another list. The firtt item in the list is the name of the variable. It is bound to the value of the second item. There are many other variables that you can set by adding them to the VARS list. Some of these variables are described in Chapter 43, and many others can be found in the lnterlis~D Reference Manual.

If you want to automatically load files, that can be done in your init file also. For'exampe, if you always want to load tho Library file SPY. DCOM, you can load it by editing tho IN|TC0MS variable to list the appropriate file in thø list starting with FILES:

12.1 YOUR NIT flu

MAKING AN IN|T FILE

(yARS iflEdlr.Llngør NIL') After
ff1LE~ _ ~PY\) Betott
Delete
Replace
Switch

()out
Undo
Find
Swap
Reprint
Edit

EddCom
Breok
Evol
Exit

FluFe 12.3. INITCOMS changed to load the file SPY.DCOM
Other files can also be added by simply adding their names to this FILES list.

Another list that can appear in a COMS list begins with "P". This list contains Interlisp-D expressions that are evaluated when the file is loaded. Do not put DEFINEQ expressions in this list. Define the function in the environment, and then save it on the file in the usual way (see Section 11.6, Page 11.7).
One type of expression you might want to see here, however, is a F0NTCREATE function (see Section 31.2, Page 31.2). For example, of you want to use a Helvetica 12 BOLD font, and there is not a fontdescriptor for it normally in your environment, the appropriate call to FOffTCREATE should be in the 'øP" list. The IN|TCOMS would look like this:

.
((VARS (D—ditLingcr NIL)) After
(FILES SPY) Betone
(~ JFoHTcREaTE (QUOTE Delcte
HEL\'ETIl',, Repace
~vyitch

1-
~juoTE _ SOL") _ .1)) ('out
Undo
Find

Swap
Reprint
Edit

EdiKom
Break
Eva
Exit

Figure 12.4. ltulTcOfl5editedtoincludeacalltofOffTCflEATE. The form will be evaluated when thelNlT file is loaded.

To quit, exit from DEdit in the usual way. When you run the function NAKEFiLES (See Section 11.6, Page 11.7.), be sure that you are connected to the directory (see Section 8.7, Page 8.4) where the INlT file should appear. Now when GREET is run, your init file will be loaded.

Y0UR INlT FILE 123

----- Next Message -----

Date: 19 Dec 91 16:48 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.164812pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11686>; Thu, 19 Dec 1991 16:48:22 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:48:12 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

r 33. MASTERSCOPE

Masterscope is a tool that allows you to quickly examine the structure of complex programs. As your programs enlarge, you may forget what variables are global, what functions call other functions, and so forth. Masterscope keeps track of this for you.

Suppose that JVTO is the name of a file that contains many of the functions involved in a complex system and that LlNTRANS is the file containing the remaining functions. The first step is to ask Masterscope to analyze these files. These files must be loaded. All Masterscope queries and commands begin with a period followed by a space, as in
ø AliLYZE FKS a Jvro

The ANALYZE process takes a while, so the 5ystem prints a period on the screen for each function it has analyzed. (See Figure 33.1)

82&. ANALYZE FNS ON 3VTO
. d.,ne
D3~. aNALY?E FNS ON LIH1R'N~

. 1a,lA.l

Figure 33.1. The Interlisp-D Executive Window affer anolyzing the files
If you are not quite sure what functions were just analyzed, type the file's CONS variable (See Section 11.5, Page 11.7.) into the Interlisp-D Executive Window. The names of the functions

stored on the file will be a part of the value of this variable.

A variety of commands are now possible, all referring to individual functions within the analyzed files. Substantial variation in exact wording is permitted. Some commands are:
ø SHoN PATHS FRDN ANY T0 ANY
ø EDIT WERE ANY CALLS functionname
ø EDIT WERE ANY USES variablename
ø Wo CALLS WDN

ø Wo CALLS functionname
ø BY WoN IS functionname CALLED
ø WD USES variablename AS FIELD

Note that the function is being called to invoke each command. Refer to the /nterlisp-D Reference Manual for commands not listed here.

Figure 33.2 shows the lnterlis~D Executive Window affer the commands ø wno CALLS GobbleDunp and ø vffo DOES JVLinScan CALL.

MASTb'R'j~OPE 331

MASTEH,COPE

NIL

7,.,'. 1,,.lillj O~LL;==: ,1)~8 iD.B~imp

(,"c.h.~t,:r~i'TJ .J;/,j~J,J .J'.'t'r'Jet'TJ J;,'~~ 1Tij Gi>"ri'.'p~" ,,)bbl,,Ffu:h ,"jbb1~'Srririll I/dump Fiji

'...,9j', "Ho clclE.. .J"i'L i '-. r, 1'"'LL
(Liri.'ci-ri 1'ø.'Cfr.3b1A 3 -h1~J
'9'A

Figure 33.2. Sample Masterscope Output

33.t The SHOW DATA command and GRAPNER
When the library package GRAPHER is loaded, (to load this package, type (FILESLOAD GRAPHER).) Masterscope's SHOWPATHS command is modified. The command will be changed to generate a tree structure 5howi ng how the program's functions interact instead of a tabular printout into the lnterlis~D Executive window. For example, typing:
ø ~ PATHS FW Proce:s—E.
produced the display shown in Figure 33.3.

.GtB.,31nT~, T L:.n;.lLl:'ø.Utøn:P,.=
~'"'-,Jt",,;r,.pj

r~infr.:p it'~ø;r.!rPr:p
r.>~>(Li;'7Uiøn~p..:
..~..JtL,;.l. Ofl'.JtLl;l.
. ',r:L,st lET _ p-:
—::J8:~inEnJ ,*.l.Tø.'r:.

ø .Err.'.r Pfl!I Pr'ni.~n,.;
:p~1y
~lnt~nlno

Figurø 33.3. SHOW PATHS Dsplay Example

All the functions in the display are part of this analyzed file or a previously analyzed file. Boxed functions indicate that the function name has been duplicated in another place on the display.

Selecting any function name on the display will pretty print the function in a window. (See Figure 33.4.)

ij.J MAsTERscopa

THE SHOW DATA COMMAND AND GRAPHER

-~&lLir1wilhS høfi
.ø~Tløo1nTwTr1no~
~9i"~

ø 1n,fl~ _ ~i~or9~~~
ø ~~&tl1r?inith. "(s
.

to~~tLisi _ ~~otLirt
~ø~.~.~rø,ø; _____
Pw:øLlrt ~(LøTTø.'

ø .' .d.~~~~;ø ~qLT~' f

ø øø..'PøintError PTl.I Frint~ninøi
ø

ø .- upv
~inlWr,r.

[LAnaPA ~propnaaoø'i (' cdttod: øø16øMAAøø3ø' L'6"døø
ø øłeCAn~'9SCorProp prcpneae (suR 1012C8L'øk])

Figutt 33.4. Browser Printout Example.

Selecting it again with the leff mOuse button will produce a dexription of the function's role in the overall system (See Figure 33.4)

~r.ø;l.1,'?U1ønø;øf'.:ø
øt1BfgøinTW:øl.riny~ _____

,p'~ø~ 1"~ø:t,-'øøø.Pr'~
. ~c.3v.Liøiniih~t

Proc&;:Eli<. _____
ø .. . Por;ø~ør' ø~:l=T7øoø.

ønf&rø.øør ~T=i Prir,t.~nir,ø
ø øPøntWr,1fl4ø

ø GerryProp i,, -
ø L-Qll:! inetAnC .rorPrtihø
1n—N—l,~ lrireFiøJ.,rningø
Din" 'pe~øbøø'c8cJ1nT;.,='øtr1n0.
i=~rMLø,ø 1rø"'ø¤øT=~',Før=Ceø'øøEtlD
ø u~' f.-ccø TO cocl

FløUrf 33.5. Browser Description Example.

33.2 Databasefns: Automatic Construction and Upkeep of a Masterscope Database

DataBaseFns is a separate library package that allows you to automatically construct and maintain Masterscope databases of your filesø The package is contained in the DATABASEFNS.DCOM file.

When DATABASEFNS.DCOM is loaded, a Masterscope database will be automatically maintained for every file whose.DATABASE

MAST'RS~OPE 333

DATABASEFNS: _ AUTOMATIC CONSTRUCTION AND _ UPKEEP OF A MASTERSCOPE _ DATABASE

property has the value YES. If this property's value is not set, you will be asked when you save the file "Do you want a Masterscope Database for this file?". Saying YES enables the DabaBaseFns to construct a Masterscope database of the file you are saving. Each time the function *AKEFILE is used on a file whose DATABASE property has a value YES, Masterscope will analyze your file and update its own database. Each file's masterscop database is kept in a separate file whose name has the form FILE. DATABASE. Whenever you load a file with a YES value for its DATABASE property, you will be asked whether you also want the database file loaded.

33.4 N~TERSCOPE
1

----- Next Message -----

Date: 19 Dec 91 16:50 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.165058pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11688>; Thu, 19 Dec 1991 16:51:02 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:50:58 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

rø 34.WHERE DOES ALL THE TIME GO?

sPY

SPY is an Interlisp-D library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

34.1 Now to use Spy with the SPY Window
The function SPY. BUTTON brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the SPY program. When you are not using SPY, the window appears as in Figure 34.1.

Figure 34.1. The SPY window when SPY 15 not being rnsed.
Ts use SPY, click either the leh or middle mouse button with the mouse cursor in the SPY window. The window will appear as in Figure 34.2, and means that SPY is accumulating data about your program.

Figure 34.2. The SPY wir.oow when SPY is being used

To turn off SPY affer the program has run, again click a mouse button in the SPY window. The eye closes, and you are asked to position another window. This window contains SPY's results. An ex~nr'ple of result window is shown in Figure 34.3.

WHERE D0ES ALL THE TIME G0' SPY 341
1

HOW TO USE SPY WITH THE SPY WINDOW

- TIrE.

l _ '3~~"H[P _ J~. _ [WIT. _ IN~&F.-1!~

17 _ '..TIrtP. PplJl'.E,-'-'..
- a ~i~~~~pT~—&-

. REPE,,TE&L'.EV~rn EJ~rn ø1 EF.—UFE

7 _ ---RR.. h.JPillU~. _ fPlLE-ø -. 4 f, IPP,9R.h 'F..n 4
Figure 34.3. The window produced afler running ¤PY

This window i5 scrollable in two directionsø horizontally, and vertically. This is useful, since the whole tree does not fit in the Winoovv. If a part that you want to see is not shown, then you can scroll the window to show the part you want to see.

34.2 How to use SPY from the Lisp Top Level
SPY can also be run while a specific function or system is being used. To do this, type the function WITH. SPY:
(VITN.sPY form)

The expression used for form should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function SPY. TREE. To do this, type:

(SPY.TREE)

The results of the last running of SPY will be displayed. If you do this, and 5PY.TREE returns (no SPY saiples have been gathQ red), your function ran too fast for SPY to follow.

34.3 Interpreting SPY's Results
Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls.
The second mode is individual. To chango the mode to individual, point to the title bar of the window, and press the middle 'n.ouse button. Choose Individual from the menu that appears. In this mode, the percentage shown is the amount of time th3t the function spent on the top of the stack.

34.2 WHERE 00E5 ALL TN5 ylMff G0? spY
1

lNTERPREn~ SPY'S RESuLtt

----- Next Message -----

Date: 19 Dec 91 16:40 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.164041pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11685>;
Thu, 19 Dec 1991 16:40:51 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:40:41 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

32. THE INSPECTOR

The Inspector is a window-oriented tool designed to examine data structures. Because Interlisp-D is such a powerful programming environment, many types of data structures would be difficult to see in any other way.

32.1 Calling the Inspector

Take as an example an object defined through a sequence of pointers (i.e. a bitmap on the property list of a window on the property list of an atom inaprogram.)
To inspect an object named NAME, type:
(IKSPECT '~)

If NAME has many possible interpretations, an option menu will appear. For example, in Interlisp-D, a litatom can refer to both an atom and a function. For example, if NAME was a record, had a function definition, and had properties on its property list, then the menu would appear as in Figure 32.1.

PRG'PS
FklS

FIELD;=~

Figure 32.1. Option Window For Inspection of NAME

If NAME were a list, then the option menu shown in Figure 32.2 would appear. The options include:
ø calling the display editor on the list;

ø calling the ~ editor (the "Typing Shortcuts",Chapter 6);
ø seeing the list's elements in a display window. If you choose this option, each element in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of numbers. (See Figure 32.3.)

ø inspecting the list as a record type (this last option would produce a menu of known record types). If you choose a record type, the items in the list will appear in the right column of the Inspector window. The left column of the Inspector window wili be made up of the field names of the record.

P~rI~.lErtr
Tr:rE'1ir.

Inip~rr

A~are'iJrd

Figure 32.2. Option Window For Inspection of Lirt

THE INSPECTOR 321

USING THE INSPECTOR

32.2 Using the Inspector

If you choose to display your data structure in an edit window, simply edit the structure and exit in the normal manner when done. If you choose to display the data structure in an inspect window, then follow these instructions:

ø To select an item, point the mouse cursor at it and press the left mouse button.

ø Items in the right column of an Inspector window can themselves be inspected. To do this, choose the item, and press the center mouse button.

ø Items in the right column of an Inspector window can be changed. To do this, choose the corresponding item in the left column, and press the center mouse button. You will be prompted for the new value, and the item will be changed. The sequence of steps is shown in Figure 32.3.
ø .

1 INPEu'T-ME-TOOl
1ie-,PErT-hl—-TQi32

a IN.u'FErT-11E-TQO3 The item in the lefl column is selected, and the middle mouse button pressed. Select the SET option from the menu that pops up.

The ev..pre:1Un re3J will be E
'/~LuQred.

cHaflGE&-'.:~"LlJ4

1]N.=~Pfi=.T-rrtE-Tc~i
2 1H".~pEcT-ttE-TI:i12

a Il You will then be prompted for the new value. Type it in.

ø6

1 [flPEQT-ME-TOOi
2 [Y.~PECT-1'1E-TOfl2

a CH~Pl,'ED-'.;~LUE The item in the right column is updated to the value of what you typed in.

Figure 32.3. The sequence of steps involved in changing a value in the right column of an Inspector window.

32.3 Inspector Example

This example will use ideas discussed in 5ection 37.1. An example, ANlMALGItAPH, is created in that section. You do not need to know the details of how it was created, but the structure

will be examined in this chapter.
If you type

(IKSPECT lI~.6liPN)

and then choosø thø Inspect option from thø menu, a display appøars as shown in Figure 32.4. ANlMAL.G~PH is being

J

33.J TkElNSPECT~

lff5PE~0R EXAMPLE

inspected as a list. Note the numbers in the left column of the inspectorwindow.

1 i’t’fI.~H ~ NIL NIL --j ’BIRD .~ NIL NIL
ø T
.i, NIL
4 NIL
5 NIL
6 NIL
? NIL
,qø NIL
9. NIL
1A. NIL
11 NIL
1~" NIL

Figure 32.4. Inspector Window For ANIMAL GRAPH, inspected as a list.
If you choose the "As A Record" option, and choose "GRAPH" from the menu that appears, the inspector window looks like Figure 32.5. Note the fieldnames in the leff column of the inspectorwindow.

UP"PH.CH"NCEL"eELFfl NIL
CR"PH. INVEP.TL~BELFN NIL
CR"PH. IFlvEp.TBCiROERFN NIL
CR"PH.FONTcH"NoEFN NIL
bRaPH.&ELETELINKFN NIL
CRaPHø~D&LINkFN NIL
URAPH.c—LETENC~UEFN HIL
bRAPH. .oo&NUGEFN NIL
oRoPH.Mo¤ENUoEFN NIL
DIREcTEDfLG NIL
o"IDE~FLo T

C.RuPHNi:DE.~ (i.’fl:H & NIL NIL --! ’BIPP & NIL GIL

Figure 32.5. Inspector Window For ANlMAL.GRAPH, inspected as an instance of a "GRAPH" record.

The remaining examples will use ANlMAL.GRAPH inspected as a list. When the first item in the Inspector window is chosen with the leff mouse button, the Inspector window looks like Figure 32.6.

1 ’ _ ø1
T
3 NIL
4 NIL
5 NIL
r~ NIL

NIL
NIL
9 NIL
1H NIL
11 NIL
1- NIL

Figure 32.6. Inspector Window For ANlMAL.GRAPH With First Element Selected
When you use the middle mouse button to inspect the seiected
list element, the display looks like Figure 32.7.

THE INSPECTOR 32 j

INSPECTOR EXAMPLE

1 ø1
T

3 NIL =
4 PIlL

5 NIL 1 iFIfl 1.19:’ 44) PIlL NIL HIL --!
’BIRD (.192 29) NIL PIlL NIL --
b’ NIL 3 (CAT (.is ,J NIL NIL NIL

PIlL j. (&UU i"1;39 7) PIlL PJIL NIL
NIL ~ ((rh,,"trtffi,,"L GJU c~T) 199 14j fiL J.IlL
9 PIlL 6 ((,,"PIIMAL ; BIRD FI.Jh) .’..~ C9. IlL
19 PIlL
11 NIL
1’.’ NIL

Figuro 32.7. Inspertor Window For ANlMAL.GRAPH and For the First Element of
ANIMALøGRAPH

How you can see that 5iX items make up the list, and you can
further choose to inspect one of these items. Notice that this is
also inspected as a list. As usual, it could also have been
inspected as a record.

Select item 5 - MAMMAL DOG CAT - with the leff mouse button.
Press the middle mouse button. Choose "Inspect" to inspect
your choice as a list. The Inspector now displays the values of the
structure that makes up MAMMAL DOG CAT. (See Figure 32.8.)

1 (h1~~MMkL GJ, lIT)
2 ilvjy’ lJ)
NIL
4 NIL
5 NIL
6 45
7

is

o i’,Do’: ClIT’,i

!) (c"’NlMlIL .~ BlRP FI."3Hj
iR, (Fi=1NTCLn"’,~r’j7R!i?e..764
ii hllIPtMlIL
12 NIL

Figure 32.8. Inspector Window for Element S From Figure 32.7 That Begins
((MAMMAL DOG CAT).

32.A THE INSPECTOR

----- Next Message -----

Date: 19 Dec 91 16:54 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.165444pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11690>;
Thu, 19 Dec 1991 16:54:53 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:54:44 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

. . . . . .

rø 34.WMERE DOES ALL THE TiME GO?

sPY

SPY is an InterlispøD library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

34.1 How to use Spy with the SPY Window
The function SPY. BUTTON brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the SPY program. When you are not using SPY, the window appears as in Figure 34.1.

Figure 34.1. The SPY window when SPY is not being used.
Ts use SPY, click either the Iefl or middle mouse button with the mouse cursor in the SPY window. The window will appear as in Figure 34.2, and means that SPY is accumulating data about your program.

sPY

Figure 34.2. The SPY wirdow when SPY is being used

To turn off SPY atter the program has run, again click a mouse button in the SPY window. The eye closes, and you are asked to position another window. This window contains SPY's results. An example of result window is shown in Figure 34.3.

WHERE D0ES ALL THE TIME Go' SPY 341
l

How TO USE SPY 'KlTH THE SPY WINDOW

rp.i)rE?
~L,:,,.*~.

IrtP.rpji=.E;";. .

øU TI~REhpYfi—&.

J!l!i .EV~fi)f. ':. ø ø1 FEPEA~OL.EU~rn -'1 EJ~J .l ER.GURE

7 _ øøø.BN.ø.F i;f;iø.iU~. _ Fpi'cføø:ø11 .. j IPP,fl~.øhJri.øii.iN
4

Fiqure 34.3. The window produced affer running SPY

Tljis window i5 scrollable in two directions, hOrizontaily, and vertically. This is useful, since the whole tree does not fit in the wiroow. If a part that you want to see is not shown, then you can scroll the window to show the part you want to see.

34.2 How to use SPY from the Lisp Top Level
SPY can also be run while a specific function or system is being used. To do this, type the function VITH SPY:
(WITH.sPY form)

The expression used for form should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function SPY. TREE. To do this, type:

(SPY.TREE)

The results of the last running of SPY will be displayed. If you do this, and SPY.TREE returns (no SPY saiples have been gathered), your function ran too fast for SPY to follow.

34.3 Interpreting SPY's Results
Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls.
The second mode is individual. To chango the mode to individual, point to the titlo bar of the window, and press the middle .~ouse button. Choose Individual from the menu that appears. In this mode, the percentage shown is the amount of time that the function spent on the top of the stack.

34.2 WHERE nQE¤ ALL THE TIME G0? SPY
1

1NTERPREfl~ SPY'S RESULTS
To look fit G iingle branch of the tree, point with the mouse curtor at one of the nodes of the tree, and press the right mouse hutton. From the menu that appeatt, choose the option SubTree. Another SPY window will appear, with just this branch of the tree in it.

Another way to focus within the tree is to remove branches from tlie tree. To do this, point to the node at the top of the branch you would like to delete. Press the middle mouse button, and choose Delete from the menu that appears.

There are also different amounts of "merging" of functions that can be done in the window. A function can be called by another function more than once. The amount of merging determines where the subfunction, and the functions that it calls, appear in the tree, and how offen. (For a detailed explanation of merging, see the Lisp Library Packages Manual.)

1

----- Next Message -----

Date: 19 Dec 91 16:59 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.165929pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11691>; Thu, 19 Dec 1991 16:59:33 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:59:29 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

tilL.. 36. FREE MENUS

Free Menu is a library package that is even more flexible than the regular menu package. It allows you to create menus with different types of items in them, and will format them as you would like. Free menus are pai~icularly useful when you want a "fill in the form" type interaction with the user.
Each menu item is described with a list of properties and values. The following example will give you an idea of the structure of the description list, and some of your options. The most commonly used properties, aiid each type of menu item will be described in Section 36.2 and Section 36.3.

36.1 An Example Free Menu

Free menus can be created and formatted automatically! It is done with the function FN. FORNATNENU This function takes one argument, a description of the menu. The description is a list of lists; each internal list describes of one row of the free menu. A free menu row can have more than one item in it, so there are really lists of lists of lists! It really isn't hard, though, as you can see from the following example:

```
(SETQ Ex~1e*anu
(F*.FORliT*EMu

'(( TYPE TITLE LABEL TitlesDonothing)
TYPE 3STATE LABEL Ex~1e3State))

( TYPE EDITSTART LABEL PressToStartEd;ting
ITEMS (EDITE*))

(TYPE EDIT ID EDITEN LABEL øø))
(*IKDDMPRDPS TITLE øEx~1e Dris Nothing))))
```

The first row has 2 items in it; one is a TITLE, and the second is a 35TATE item. The second row also has 2 items. The second, the EDIT item, is invisible, because its label is an empty string. The caret will appear for editing, however, if the EDlTSTART item is chosen. Windowprops can appear as part of the description of the menu, because a menu is, affer all, just a special window. You can specify not only the title with WINDOWPROPS, but also the position of the free menu, using the "Ieff" and "bottom" properties, and !he width of the border in pixels, with the "border" property. Evaluating this expression will return a window. You can see the menu by using the function OPENW. The following example illustrates this:

FREE MENUS 361
1

AN EXAMPLE FREE MENU

6i,'~T~ E;mD.1c1d~nij

.;F(,7,fJp,[4,,¤~]~fJ.J\J ,, ø. T' Gf TITLE LBEL T ir1~,flN~rr T
.T"rE =-ø'T."TE L"bEL E:.Jm"1c5tJcs'!'.
. 'FE =,IT:..THF:T

LEL =r~'=Tu"'r~t'tEditing
lTEfl= cOlTEN'

T"E 'IT ID EDITEm L~8EL ,
'"..ililci=,.."=cipT=..

TITLE ' ;,.,1";c Din' 1'luthlnj.'?

.TT9'i i)pf)liff molMertiJ'i
f.hi1ltlDu',V'r#' j64

Figure 36.1. An example free menu

The next example shows you what the menu looks like affer the EDlTSTART item, PressToStartEditing, has been chosen.

T,r f~"=.Oi=i1'1,=irhin3 E':,mp1~,='.=r.,r~
P~'~='"TJT..,rTEJ1r1r1.j A

Figure 36.2. Free menu affer the EDlTSTART item has been chosen
The following example shows the menu with the 3STATE item in its T state, with the item highlighted (In the previous bitmaps, it was in its neutral state.)
.
ø c l 1 1

.1-=':.OiJ-tljrhini=!
,:'T'='Ot..;rrE'liriiJ,

FigUre 36.3. Free menu with the 35TATE .tem in its T state
Finally, Figure 36.4 shows the 35TATE item in its NIL state, with a diagonal line through the item

T1r le.".OcNorhing E..::r'~ _ 1 _ = _ ø'.'..,i.~
Rrn. ;", T,St.arrEdir,iri,

. . . . .

Figure 36.& Free menu with the 3STATE item in its NIL state
If you would like to specify the layout yourself, you can do that too. See the Lisp Library Packages Manual for more information.

36.2 Parts of a Free Menu Item
There are 8 different types of items that you can use in a free menu. No matter what type, the menu item is easily described by a list of properties, and values. Somo of the properties you will use most often are:

36.2 FREE MENUS
1

PARTS OF A FREE MENU ITEM

LABEL Required for every type of menu item. It is the atom, string, or bitmap that appears as a menu selection.

TYPE One of eight types of menu items. Each of these are described below.

MESSAGE The message that will appear in the prompt window if a mouse button is held down over the item.

ID An item's unique identifier. An ID is needed for certain types of menu items.

ITEMS Used to list a series of choices for an NCHOOSE item, and to list the ID's of the editable items for an EDITSTART item.
SELECTEDFN The name of the function to be called if the item is chosen

36.3 Types of Free Menu Items
Each type of menu item is described in the following list, including an example description list for each one.

Momentary This is the familiar sort of menu item. When it is selected, the function stored with it is called. A description for the function that creates and formats the menu looks like this:
(TYPE WEKTARY
LABEL Blink-K-Rin9

*ES~6E øBlinks the screen and rings bellsø
s—LEcTEDFK RIKBELLS)

TOGGL— This menu item has two states, T and NIL. The default state is NIL, but choosing the item toggles its state. The following is an example description list, without code for the SELECTEDFN function, for this type of item:
(TYPE T~6LE
LABEL hi~isab1e

sELEcTEDFN changeIl*State)

3STATE This type of menu item has 3 states, NUIETRAL, T, AND NIL Neutral is the default state. T is shown by highlighting the item, and NIL is shown with diagonal lines. The following is an example description list, without code for the SELECTEDFN function, for this type of item:
(TYPE 3STATE

LABEL correctpograøAllofflospelling
sELEcTEDFli ToggleSpellingcorrection)

TITLE This menu item appears on the menu as dummy text. It does nothing when chosen. An example of its description:
(TYPE TITLE LABEL øChoices:")

NWAY A group of items, nnly one of which can be chosen at a time. The items in the NWAøY group should all have an ID field, and the ID's should be the same. For exan1Fle, to set up a menu that would allow the user to chose betvveei Helvetica, Gacha, Modern, and Classic fonts, the descriptions might look like this (Once again, without the code for the SELECTEDFN):
(TYPE lAY ID F~Tc~Ic'
LABEL blvetica
sELEcTEDFN changeFont)

FREE MENUS 36)

I

TYPES OF FREE MENU ITEMS

(TYPE NVAY ID FOQTCKICE
LABEL Gacha
SELECTEDF

(TYPE lAY ID F05TliCriC0ha,~n8efont)
LABEL Modern

SELECTEDFli Chan2eFont)
(TYPE KAY ID fONTCHOIC
LABEL Classic

SELECTEDFN Changefont)

NCHOOSE This type of menu item is like NWAY except that the choices are given to the user in a submenu. The list to specify an NCHOOSE menu item that is analogous to the NWAY item above might look like this:
(TYPE MC~SF
LABEL FontChoices

ITEMS Helvotica Gacha Modern Classic)
SELECT DfK Changefont)

EDlTSTART When this type of menu itein is chosen, it activates another type of item, an EDIT item. The EDIT item or items associated with an EDlTSTART item have their lD's listed on the EDlT5TART's ITEMS property. An example description list is:
(TYPE EDITSTART LABEL øFunction to add? ITEMS (Fn))
EDIT This type of menu item can actually be edited by you. It is often associated with an EDlT5TART item (see above), but the caret that prompts for input will also appear if the item itself is chosen. An EDIT item follows the same editing conventions as editing in Interlisp-D Executive window:
Add Characters by typing them at the caret.
Move the caret by pointing the mouse at the new position, and clicking the leff button.

Delete Characters from the caret to the mouse by pressing the right button of the mouse. Delete a character behind the caret by pressing the back space key.

Stop editing by typing a carriage return, a Control-X, or by choosing another item from the menu.
An example description list for this type of item is:
(TYPE EDIT ID Fn LABEL øø)

36.4 FREENEMus
1

----- Next Message -----

Date: 19 Dec 91 17:05 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.170545pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11694>;

Thu, 19 Dec 1991 17:05:54 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:05:45 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

37. THEGRAPHER

37.1 Say it with Graphs

Grapher is a collection of functions for creating and displaying graphs, networks of nodes and links. Grapher also allows you to associate program behavior with mouse selection of graph nodes. To load this package, type
(FILESL~ GliPHER)

Figure 37.1 shows a simple graph.
i ’iLk w.F."PH ’N M’L.l;R"PH ’NlM’øL r;P"Pffø’
,(.h,,lINGUY!:łw’,1513.y1
14’

-FIH

. NIM"L, BIRO

Figure 37.1. A Simple Graph

In Figure 37.1 there are six nodes (ANIMAL, MAMMAL, DOG, LAT, FISH, and BIRD) connected by five links.
A GRAPH is a record containing several fields. Perhaps the most important field is GRAPHNOD—S - which is itszlf a list of GRAPHNODE records. Figure 37.2 illustrates these data structures The window on top contains the fields from the simple graph. The window on the bottoms an inspection of the node, DOG.

SAY lTWlTH GRAPHS

i9’1, I ET ø’NI1’1,,L.CR~PH’.i
Ilvl,l = ’-#"=9,1j~’j’,’3

GPPH.cr"ilLEL,,ø’BELFN ’IL
ø 1’R"pH. Ili’ø!ERTLBELFN 1.lIL
ø H. Ifløø.øøERTBDPDEPFN tilL
H.FGtTi’.HNoEFPl 1ølIL

, rPø’PH,t.lL/EløllDEFbl IL
. OIRECTECFLi, (ilL

ø rp..’Pflbll) ~ I.F = , tilL III ’.øB.IP.D NIL ff IL

. NOOEBOPOER ’ilL
tiODEL,,’BEL loo
, , ’tODEFONT ~’FOIiT ’
. .OtlffO&EO- It’lL -

. t,iODE’.~,lOTH ’.4,
, IiUOEL6EL.’-H~OE øølIL
, NODELfiELBlTIrt,iøP øIIL
, I,iUDEPUITlClI.l in
NODE ID 300

Figure 37.2. Inspefling a Graph and a Node

The GRAPHNODE data structure i~ described by its text (NODElD), what goes into it (FROMNODES), what leaves it (TONODES), and other fields that specify its looks. The basic model of graph building is to create a bunch of nodes, then layout the nodes into a graph, and finally display the resultant graph. This can be done in a number of ways. One is to use the function NODECREATE to create the nodes, LAYOUTGRAPH to lay out the nodes, and SHOWGRAPff to display the graph. The primer shows you two simpler ways, but please see the Library Packages Manual for more information about these other functions. The primer's first method is to use SHOWGRAPH to display a graph with no nodes or links, then interactively add them. The second is to use the function LAYOUT5EXPR, which does the appropriate NODECREATES and a LAYOUTGRAPH, with a list.
The function SHQWGRAPH displays graphs and allows you to edit them. The syntax of SHOWGffAPH is

(~liPH graph window lefibuttonfn middlebuttonfn
topjustifffflg alloweditflg copybuttoneventfn)
Obviously the graph structure is very complex. Here's the easiest way to create a graph.
~.6liPN III)

lS5~liPN P.6liPH ø5Y Sraphø KIL NIL NIL T)

Figurø 37.3. My Graph

37.2 THEGRAPHER .J

SAY IT WITH GRAPHS

You will be prompted to create a small window as in Figure Figure 37.3. This graph has the title My Graph.
Hold down the right mouse button in the wiridow. A menu of graph editing operations will appear as in Figure 37.4.

D;Ier~ Link
&=h~~n9e ib P.I
ljbel g,nill~.r
l&'bel l~.roø~.r
Dir~..ct~.i1
SIdPg
~ BoiødP.r

'h;~d"
'Tr"P

Figure 37.4. A Menu of Graph Editing Operations
Here's how to use this menu to:

Add a Node Start by selecting Add Node. Grapher will prompt you for the name of the node (See Figure 37.5.) and then its position.

Figure 37.5. Grapher prompts for the name of the node to add affer Add Node is chosen from the graph editing menu.

Position the node by moving the mouse cursor to the desired location and clicking a mouse button. Figure 37.6 ,hows the graph with two nodes added using this menu.

~irør-ri~tle
s~~'ondnod~

Figure 37.6. Two nodes added to MY GRAPH using the graph ed it.q.g menu
AddaLink Select Add Link from the graph editing menu The Prompt
window will prompt you to select the two nodes to be linked.
(See Figure 37.7.) Do this, and the link will be added.

o .

first-node
,.ccond-node

Figure 37.7. The Prompt window will prompt you to select the two nodes to link.

THEGRAPHER 37.3

SAY IT WITH GRAPHS

DeleteALink Select Delete Link from the graph editing menu. ThePrompt
window will prompt you to select the two nodes that should no
longer be linked. (See Figure 37.8.) Do this, and the iink will be
deleted.

r _’rr-n>’;1~
;’~"or,’j-nod;

FigUre 37.8. The Prompt window will prompt you to Seje~~ ’.1 ryo nodes that
shouid no longer be linked.

Delete A Node Select Delete Node from the graph editing menu. The Prompt
window will prompt you to select the node to be aeleted. (See
Figure 37.9.) Do this, and the node will be deletea.

firs. r-nod"
,L’0fl’S1-fl0d~

Figure 37.9. The prompt to delete a node

Moving a Node Select "Delete Node" from the graph editng menu. Choose a
node pointing to the it with the mouse cursor, and pressing and
holding the leff mouse button. When you move the mouse
cursor, the node will be dragged along. When the node is at the
new position, release the mouse button to deposit the node.

The commands in this menu are easy to learn. Experiment with
them!

37.2 Making a Graph from a List
Typically, a graph is used to display one of your program’s data
structures. Here is how that is done.

LATOUTSEXPR takes a list and returns a GRAPH record. The
syntax of the function is

(UYWTSEXPR sexpr format ~xing font motberd
penonald fam;lyd)
For example:

(u~T10Q AKIliL.TREE ’(MIlL (l’~ ~ CAT) BIli FISH))
AaIliL.6liN

37.4 THEGRApHER

MAKING A GRAPH FR0M A LIST

b~YouTSE*PR AKIliL .TREE øHoRIZ0NTALi~)

(Eli N AHIliL.GliPN øNj Grøpbø NIL KIL a T)

This is how Figure 37.1 was produced.

37.3 Incorporating Grapher into Your Program
The Grapher is designed to be built into other programs. It can call functions when, for example, a mouse button is clicked on a node. The function SHOWGRAPff does this:

(~liPH graph window leflbuttonfn middlebuttonfn
topjusti~Rg alloweditflg copybuttoneventfn)

For example, the third argument to SHOWGRAPH, leftbuttonfn, is a function that is called when the lefl mouse buttoii 15 pressed in the graph window. Try this:
(DEFIK—Q (~.LEfT.BUTT0N.FUNCTI0N
(TNE.6liPHNooE THE.GliPH.wIN~)
(INSPECT TNE.6liPNNooE)))

(~liPH FlILY.61PN øInspoct&bla fiilyø
(F~TIK N".LEFT.BUTTa.FuNCTIo*)
liIL NIL T)

In the example above, liT.LEFT.BUTTON. FUNCTION simply calls the inspector. Note that the function should be written assuming it will be passed a graphnode and the window that holds the graph. Try adding a function of your own.

37.4 More of Grapher

Some other Library packages make use of the Grapher. (Note: Grapher needs to be loaded with the packages to use these functions.)

ø NASTERSCOPE: The Browser package modifies the Masterscope command, . SHOW PATHS, so that its output is displayed as a graph (using Grapher) instead of simply printed.
ø GRAPHZOOM: allows a graph to be redisplayed larger or smaller automatically.

THEGRAPHER 375

----- Next Message -----

Date: 19 Dec 91 17:11 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.171147pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11697>; Thu, 19 Dec 1991 17:11:55 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:11:47 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

41. RESOURCE MANAGEMENT

41.1 Naming Variables and Records
You will find times when one environment simultaneously hosts a number of different programs. Running a demo of several programs, or reloading the entire Interlisp-D environment from

floppies when it contains several different programs, are two examples that could, if you aren't careful, provide a few problems. Here are a few tips on how to prevent problems:
ø If you change the value of a system variable, ffENUHELDVAIT for example, or connect to a directory other than (DsK)<LlsPFlLEs>, write a function to reset the variable or directory to its original value. Run this function when you are finished working. This is especially important if you change any of the system menus.

ø Don't redefine Interlisp-D functions or CLl5P words. Remember, if you reset an atom's value or function definition at the top level (in the Interlisp-D Executive Window), the message (Some.Crucial.Function. Or. Variable redefined), appears. If this is not what you wanted, type UNDO immediately!

If, however, you reset the value or function definition of an atom inside your program, a warning message will not be printed.
ø Make the atom names in your programs as unique as possible. To do this without filling your program with unreadable names that noone, including you, can remember, prefix your variable names with the initials of your program. Even then, check to see that they are not already being used with the function BOUNDP. For example, type:
(~P øB&ckgroundhnu)

This atom is bound to the menu that appears when you press the leff mouse button when the mouse cursor is not in any window. BOUKDP returns T. BOUNDP returns NIL if its argument does not currently have a value.

ø Make your function names as unique as possible. Once again, prefixing function names with the initials of your program can be helpful in making them unique, but even so, check to see that they are not already being used. GETD is the Interlisp-D function that returns the function definition of an atom, if it has one. If an atom has no function definition, GETO returns NIL. For example, type:
(GEffl 'CAR)

A non-NIL value i~ returned. The atom CAR already has a function definition.

ø Use complete record field names in record FETCHes and REPLACEs when your code is not compiled. A Complete record field name is a list Consisting of the record declaration name and the field name. Consider the following example:
REC0RD N~ FIRST LAST))

SETQ Nyfflrn create Nl— FIRST 'John LAST '~ith))
FETCH (~ FIRST) OF Mylrn)

ø Avoid reusing names that are field names of Interlisp-D System records. A few examples of system records follow. Do not reuse these names.

RECORD RE6IOl (LEFT RoTTOl WIDTH NEIGHT))
RECORD POSITIK xC00RD
RECORD Ili6E~ BITliYCP00RD)))

ø When you select a record name and field names for a new record, check to see whether those names have already been used.

Call the function RECLOOK, with your record name as an aørgument, in the lnterlis~D Executive Window. (See Figure 41.1.) If your record name is already a record, the record definition will be returned; otherwise the function will return NIL.

- . 11

4..Oø:ø(fi.ECL)OY~ FB;'~ITiON)
!'P\ECCPO
PO~1TI)N

[øT\L~'.lp:E)F;,P "So'1P~D!
(8Nfl (LlSTP O~TUM\
(NUl18—P~P !',C4l'~ D~TUfi1:))

(i\"('~'T—h1\\j (NUMBER— (CDR OurOIl]
5ik(P~ECLOUff N~,~P~~)
NIL
5~'~E

Figuvø 41.1. RECLOOK returns tbe record definition If ts argument is already declared as a record, NIL otne~ise.

Call the function FIELDLOOK with your new field name in the InterlispøD Executive Window. (See Figure 41 2.) If your field name is already a field name in another record, the record definition will be returned; otherwise the function will return NIL.

412 RESOURCE MANAGEMENT ø1

NAMING VARIABLES AND RECORDS
, 1
~.4'+(fIELOLOOft Y96COORD)
((RECORD
pO'e.ITION

(:~'COORO \COORD)
[T'Y'PE"ø (~NP (LIvt~TP O4TUbl!
(NUblBERP (CAR D,iTUhl\
I:.NUMBERP ("OR D~TUftt]
(S\ø'~~.TEb1)\)

55~(FIELPLOPh .ip~;\
NIL
58-

Figure 41.2. FIELDLOOK returns the record definition if ItS argument Is already the field ofarecord.NILothe~ise

41.2 Some Space and Time Considerations
In order for your program to run at maximum speed, you must efficiently use the space available on the system. The following section points out areas that you may not know are wasting valuable space, and tips on how to prevent this waste.
Often programs are written so that new data structures are created each time the program is run. This is wasteful. Write your programs so that they only create new variables and other data structures conditionally. If a structure has already been

created, use it instead of creating a new one.
Some time and space can be saved by changing your RECORD and TYPERECORD declarations to DATATYPE. DATATYPE is used the same way as the functions RECORD and TYPERECORD. (See Chapter 24.) In addition, the same FETCH and REPLACE commands can be used with the data structure DATATYPE creates. The difference is that the data structure DATATYPE creates cannot be treated as a list the way RECORDs and TYPEREC0RDs can.

41.2.1 Global Variables

Once defined, global variables remain until Interlisp-D is reloaded. Avoid using global variables if at all possible!
One specific problem arises when programs use the function 6ENSYff. In program development, many atoms are created that may no longer be useful. Hints:
ø Use

(DELDEF atomname 'PKP)
to delete property lists, and
(DELDEF atomname 'vARS)

RESOURCE MANAGEMENT 413

----- Next Message -----

Date: 19 Dec 91 17:15 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.171603pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11560>; Thu, 19 Dec 1991 17:16:06 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:16:03 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

SOME SPACE AND TIME CONSIDERATIONS

to have the atom act like it is not defined.

These not only remove the definition from memory, but also change the appropriate f 11 eCOffS that the deleted object was associated with so that the file package will not attempt to save the object (function, variable, record definition, and so forth) the next time the file is made. Just doing something like
(SETQ (arg at~nm) '~IE)

looks like it will have the same effect as the second DELDEF above, but the SETQ doesn't update the file package.
ø If you are generating atom names with GENSYN, try to keep a list of the atom names that are no longer needed. Reuse these atom names, before generating new ones. There is a (fairly large) maximum to the number of atoms you can have, but things slow down considerably when you create lots of atoms.
ø When possible, use a data structure such as a list or an array, instead of many individual atoms. 5uch a structure has only one pointer to it. Once this pointer is removed, the whole Structure will be garbage collected and space reclaimed.

41.2.2 Circular Lists

If your program is creating circular lists, a lot of space may be wasted. (Note that many cross linked data structures end up having circularities.) Hints when using circular lists:
ø Write a function to remove pointers that make lists circular when you are through with the circular list.

ø If you are working with circular lists of windows, bind your main window to a unique global variable. Write window creation conditionally so that if the binding of that variable is already a window, use it, and only create a new window if that variable 15 unbound or NIL.

Here is an example that illustrates the problem. When several auxilIary windows are built, pointers to these windows are usually kept on the main window's property list. Each auxilIary window also typically keeps a pointer to the main window on its property list If the top level function creates windows rather than reusing existing ones, there will be many lists of useless windows cluttering the work space. Or, if such a main window is closed and will not be used again, you will have to break the links by deleting the relevant properties from both the main window and all of the auxiliary windows first. This is usually done by putting a special CLOSEFli on the main window and all of its auxiliary windows.

41.2.3 When You Run Out Of Space
Typically, if you generato a lot of structure! that won't get garbage collected, you will eventually run out of space. The important part ii being aNe to track down those structures and

4I.4 REsouRcE MANAGEMENT
I

SOME SPACE AND TIME CONSIDERATIONS

the code that generates them in order tO become more space efficient.

The Lisp Library Package GCHAX.DCOM can be used to track down pointers to data structures. The basic idea is that GCHAX will return the number of references to a particular data structure.

A special function exists that allows you to get a little extra space 50 that you can try to save your work when you get toward the edge (usually noted by a message indicating that you should save your work and sysin a fresh Lisp). The GAINSPACE function allows you to delete non-essential data structures. To use it, type:

(liIKSPACE)

into the Interlisp-D Executive Window. Answer "N" to all questions except the followi ng.
ø Delete edit history
ø Delete history list.

ø Delete values of old variables.
ø Delete your MASTERSCOPE datadase
ø Delete information for undoing your greeting.
Save your work and reload Lisp as soon as possible.

RESOURCE MANAGEMENT 41 S

----- Next Message -----

Date: 19 Dec 91 17:23 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.172334pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11625>;
Thu, 19 Dec 1991 17:23:37 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:23:34 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

42. SIMPLE INTERACTIONS WITH THE CURSOR, A BITMAP, AND A WINDOW

The purpose of this chapter is to show you how to build a moderately tricky interactive interface with the various lnterlis~D display facilities. In particular how to move a large bitmap (larger than 16 x 16 pixels) around inside a window To do this, you will change the CURSORINFN and CURSOROUTFN properties of the window. If you would also like to then set the bitmap in place in the window, you must reset the BUTTOKEVENTFN. This chapter explains how to create the mobile bitmap.

42.1 An Example Function Using GETMOUSESTATE

One function that you will use to "trace the cursor" (have a bitmap follow the cursor around in a window) is GETNOUSESTATE. This function finds the current state of the. mouse, and resets global system variables, such as LASTM0USEX andLASTMOUSEY.

As an example of how this function works, create a window by typing

(SETQ EzMPLE.wIN~ (CREATEI))

into the Interlisp-D Executive window, and sweeping out a window. Now, type in the function
(DEFIKEQ (PRIKTC00RDS (V)

P~TPRI*T ø(ø ~TuouSEx ., . US~EY ø)ø)
BLocK)

6E~SESTATE)))

This function calls GETNOUSESTATE and then prints the new values of LASTNOUSEX and LASTMOUSET in the promptwindow. To use it, type

(WIKraoPRoP EXlPLE .ilI*~ 'CURSD~EDFK 'PRIaTC00RDS)
The window property CURSORffOVEDFN, used in this example, will evaluate the function PRINTCOORDS each time the cursor is moved when it is inside the window. The position coordinates of the mouse cursor will appear in the prompt window. (See Figure 42.1.)

SIMPLE INTERACTIONS M'ITH THE cuRsoR. A BlTMAP, AND A WINDOW 42 f
I

AN EXAMPLE FUNcTION USING GETMOUSE5TATE

Figure 42.1. The current position coordinates of the mouse cursor are shown in the prompt window

42.2 Advising GETMOUSESTATE
For the bitmap to follow the moving mouse cursor, the function GETKOUSESTATE is advised. When you advise a function, you can add new commands to the function without knowing how it is actually implemented. The syntax for advise is

(RISE fn when where what)

fn is the name of the function to be augmented.
when and where are optional arguments. when specifies whether the change should be made before, affer, or around the body of the function. The values expected are BEFORE, AFTER, or AROUND.

what specifies the additional code.

In the example, the additional code, what, moves the bitmap to the position of the mouse cursor. The function GETNOUSESTATE will be ADVISEd when the mouse moves into the window. This will cause the bitmap to follow the mouse cursor. ADVISE will be undone when the mouse leaves the window or when a mouse button is pushed. The AOVISEing will be done and undone by changing the CURSORINFK, CURSOROUTFN, and BUTTONEVENTFK forthewindow.

42.3 Changing the Cursor

Ofif laot part of tho examplo, to give thø impression that a bitmap is dragged around a window, thø original cursor should disappear. Try typing:
(CURSOR (CURSORCR—Rrt (6I~PCREAtt 1 l) 1 11

ø2.2 SINPU NVE~CYI0NS WITH Tn: CURSOR. A StTNAP. ANO A WIHIOOW

CHANGING THE CURSOR

into the lnterlis~D Executive Window. This causes the original cuttor to disappear. It reappears when you type
(CURSOR T)

When the cursor is invisible, and the bitmap moves as the cursor moves, the illusion is given that the bitmap is dragged around the window.

42.4 Functions for "Tracing the cursor"
To actually have a bitmap trace (follow) the cursor, the environment must be set up so that when the cursor enters the tracing region the trace is turned on, and when the cursor leaves the tracing region the trace is turned off. The function

Establish/Trace/Data will do this. Type it in as it appears (note: including the comments will help you remember what the function does later).

(DEFIKEQ Establish/Trace/rata

```
[LANR0 (ønd tracebiteap cursor/rightoffse~ cursor/heighteffse~ GCGA6P)

ø This functlri is collød to ostablish tha døti to tracø
the døsirod bitaapø øøndø is the øind~ in øhich the tracing
is to take place, łtracebitøap" is the

øcursor/rightoffsetø and ø~~~50~'~~~g~~0t~~~~~5~i~~1 b~~i~~t~g~~5
øhich dete~ine the hotspot of the tracing biteap.
As "cursor/heightof'set and øcursor/rightoffsetø increase
the cursor hotspot :ves up and to the right.
If GCGAGP is non-NIL, GcGAC øill be disabled.)

(PRoG NIL

(if (0R NULL ønd)
(NULL tracebitaap))

then (PLAYTUN— (LIST (CONS 1000 4000)))

(if ~&~&pRET~RN))
then (GC6A6))

ø Create a blank cursor.)

(SSEETTQQ
~8BrnUNNKKCTliURCS0ECRtiR(soBIRllNAø(CPCURRsoEARTcEREl,eT~loø)jwxc~~2~øj~
ø ø Set the CURSOR IN and 0UT FNS for ønd to the
Jolloeing:)

(*INroNPRoP ønd UTE CURSORINF
(FU TIrn SETUP/Tlic

(WINDoNPooP ønd~~TE CURSoRoUTFENNJJ
(FU TIoN UNTlic—/CURSOR))

(O ø To all",' the bita,ap to be set den in the øindw bY
pressing a '"ouse button, include this line.
0ther',,'ise, it is not needed)

(WINnoNPRop ønd (UTE RUTToNEVENTFN)
(FUNCTIoN PLACEIBITNAPIINIwINrGN))

Set up Global Variables for the tracing 9eratien)
(SETQ øTRAcElITNApø tracebiteap)
TQ øRIGNTTliCE'oFFSETø(oR cursor/rightoffse~ 0)

5>sfE~TQ øHEIGHTTRACEIoFFSETø(0R cursor/hei htoffset ))
TQ ø0LQBIT~PPosITIoNø(BIfflPCREATE llNArnIOTN tracebitøap)

(SETQ øTliCCwfNDoNø rndj)) BITNApHEI6hT tracebiteap)))

SIMPLE INTERAcT'oNs WITH THE cuRsoR. A BJTMAP. AND A WINDOW 423
1

FUNCTIONS FOR "TRACING THE CURSOR'

When the function Establish/Trace/Data is called, the
functions SETUP/TRACE and UNTRACE/CURSOR will be installed
as the values of the window's WlNDOWPROP¤, and will be used
to turn the trace on and off. Those functions should be typed in,
then:

(DEFINEQ SETUP/TRACE
```

```
[ADA ønd)

(O ø This function is and's cuRSORIKFK.
It siepiy resoti thø last trace position and the current
tracing region. It also raadvises fiETNouSESTATE to perforn
the trace function after each call.)

(if øTRAcEBITNAPø

then SETQ łLAST-lPACE-XPO5ø -zOo
SETQ øLAST-TRACE-YPOSø -zoooł)

SETQ øvNDREGIaø (WINOaNPROP and (ATE REGI0N)))
WIN~flROP and (ATE TliCIK)
T)

ø :ake the cursor disappear)

CURSOR łBLANKTRACECURSORø)
ANISE QUOTE GEThOUSESTATE)
QUOTE AFTER)
IL

(QUOTE (TPACE/CURSOR]))

(Dt~EQ(UaNTRACE/CURSOR

ø Th1s function is ønd's CURsOROUTFN.
The function first checks if the cursor is currently being
traced; if so, it replaces thø tracing biiaap aith ahat is
under it and then turns tracing off by unadvislng
6ETNOUSESTATE and setting thø TliCIK ainda propertj of
łTRACEWINOoOø TO NIL.)

(if (VIN~PnOP øTRACEWINOONł(QUOTE TliCIK))
then (BITBLT ø0LOBITNAPPOSITIONø o o (scREENBIllNAP)
IPLUS CAR łffDRE6IONø)øLAST-lRACE-xPOSø)
(IPLUS 1CADR ø:DREGIOffo )øusT-TlicE-YPosø))
(WINOoePRoP ølliCEMINOONø(QUOTE TRACIK)
NIL))

replace the original cursor shape)
(CURSOR T)

unadvise 6E~sESTATE)
(U~'"ISE (QUOTE 6ETNOUSESTAlE]))

The function SETUP/TRACE has a helper function that you must
also type in. It is TRACE/CURSOR:
(DEFINEQ (TliCE/CURSoR
[LANRli NIL

ø This functi: does thø actual BITBLTln of thø tracing
blteap. This functla Is cølled after a GE f TATE, abl ø
tracing.)

(PRoG ( xpos IDIFFERENCE LAsTNOUsEZ øTRACEWINnoNł øRIGNTTRACE/OFF
[ypoo IDIFFERENCE LAsTNOUsEY øTRACE*INr~ł øNEIGnTTRACE/OfFSsEETiJ)))
ø If there Is an ørror In thø function, ress thø riKbt
button to unodvlsø thø function. This øill eep thø ac inø
fr: locking up.)

(If (LASTNOUSESTATE RIGiiT)
```

(if ~t1h~~~ (NUNAPuISE (QUOTE 6ElS~5ESlATE)))
Q xpoa øLAST-TRACE-XP0sø
(KEO ,pea øLAsl-TeACE-YPOSøj)
thøn

ø Restoro ahøt ~s undør the eld pooltla of thø trøcø
OilUp)

(SITGLY øOLliIllnApposITIGøø o o (IREE5IlliA~)

ø2A SIMPLE lNTE~CJ\OHS WITH THE CURSOR. A BtTMAP. AflO A WINDOW
1

F,
FUNCTIONS FOR "TRACING THE CURSOR"

IPLUS CAR øil

IPLUS CADR ø:DDRRESEGISIrnø2øLASTTRAC

ø søvø øb&t ø111 bø undør thø position of thø nee trøcø
biteap)

(51 TILT SCREENBITNAP)
[IPLUS (CAR øaDREGIaø)
xpos)

(IPLUS øvNDREGIoaø O O)

BIfflLT the tracG blt:p onto thø nøø position of thø
eouse.)

(8ITBLT øTRACEBITNAPł O O ~5CREENBITNAP)
(IPLUS (CAR øilDRE ION")

(fPLUS (CADS øffORE6IONø)
ypos

NIL NIL YE INPUT)

(ONDTE P liT))

ø Savu the current position as the last trace position.)

(SETQ øLAST-TRACE-xPDSø xpos
(SETQ "LAST-TRACE-YPOSł ypos

The helper function for UHTRACE/CURSOR, called
UNDO/TRACE/DATA, must also be added to the environment:
(DEFINEQ (UNDo/TRACE/DATA
[LISA NIL

ø The purpose of this function is to turn tracing off and
to free up the global variables used to trace the bitaap, so
that thej can be garbage collected.)

Check if the cursor is currently being traced.
It so, turn it off.)

UiTRACE/CURSoR)

WINDoNPRDP łTliCE*IN~ł(uTE CURSDRINFN)
NIL)

```
(WINDo*PRDP łTRACEwINDDNł(uTE CUR~R0UTFN)
NIL)

SETQ "TRACEBITsAPł NIL)
SETQ øRIGNTTlicE/oFFsETø NIL)
SETQ øHEIGHTTRACE/OFFSETø NIL)
SETQ øOLDBITliPP0SITIDNø NIL)
SETQ łTRACE*I~ł NIL)

ø Turn GCGAG on)

(6C6A6 TJ))
```

Finally, if you inCluded the WlNDOWPROP to allow the user to place the bitmap in the window by pressing a mouse button, you must also type this function:

```
(D[E~DEAQ, ønd)
UNADVISE (SETNDUSESTATE))

fBITBLT øTliCEBITNAPø O O SCREENBIlNAP)
(IPLUS (CA 0Nø)
xpo

(IPLUS (CADR øiDREGIONø)
ypos)

NIL NIL (UTE INPUT)
(ATE PAINT]
```

That's all the functions!

SIMPLE INTERAcTioNs NITH THE cuRsoR, A BITMAP, AND A WIND0W 42 S

p

RUNNING THE FuNcTlGh5

42.5 Running the Functions

To run the functions you just typed in, first set a variable to a window by typing something like
(SETQ EXMPLE.wIN~ (CR—ATEI))

into the Jnterlisp-D Executive window, and sweeping out a new window. No'rv, set a variable to a bitmap, by typing, perhaps,
(SETQ ExlPLE.BTn (—DITl))
Type

(Estab1isfl'Trsce'Oo~ EXlPLE.WIN~ EXlPLE.BTK))
When you move the cursor into the window, the cursor will drag the bitmap.

(Note: If you want to be able to make menu selections while tracing the cursor, make sure that the hotspot of the cursor is set to the extreme right of the bitmap. Otherwise, the menu will be destroyed by the BITSLTs of the trace functions.)
To stop tracing, either

ø move the mouse cursor out of the window;
ø press the right mouse button;
ø call the function UNTRACE/CURS0R.

u.6 SIMPLE lNTEPACT~NS WITH THE CURSOR. A SITMAP. AND A WlN00W

----- Next Message -----

Date: 19 Dec 91 17:30 PST
From: sybalsky:PARC:Xerox
To: sybalsky
Message-ID: <<91Dec19.173105pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11702>; Thu, 19 Dec 1991 17:31:18 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:31:05 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers---->

r fi. OTMER R—FERENCES THAT WILL
8E USEFUL TO YOU

Here are some references to works that will be useful to you in addition to this primer. Some of these you have already been referred to, such as:

ø The InterlispøD Reference Manual
ø The Library Packages Manual
ø The User's Guide to SKETCH
ø Thell86orllO8User'sGuide

In addition, you can learn more about LISP with the books:
ø Interlisp-D: The languago and its usage by Steven H. Kaisler. This book was published in 1986 by John Wiley and Sons, NY.

ø Essential LISP by John Anderson, Albert Corbett, and Brian Reiser. This book was published in 1986 by Addison Wesley Publishing Company, Reading, MA. It was informed by research on how beginners learn LISP.

ø The Little Lisper by Daniel P. Friedman and Matthias Felleisen. The second edition of this book was published in 1986 by SRA Associates, Chicago. This book is a deceptively simple introduction to recursive programming and the flexible data structures provided by LISP.

ø LISP by Patrick Winston and Berthold Horn. The second edition of this book was published in 1985 by the Addison Wesley Publishing Company, Reading, MA.

ø LISP: A Gentle Introduction to Syabolic Coaputation by David S. Touretzky. This book was published in 1984 by the Harper and Row Publishing Company, NY.

Finally, there are three articles about the Interlisp Programming environment:

ø Poaer Tools Tor PrograffaersbyBeauSheil. It appeared in Datamation in February, 1983, Pages 131 - 144.
ø The Interlisp Prograffaing Environaent by Warren Teitelman and Larry Masinter. It appeared in April, 1981, in lEEE Computer, Volume 14:1, Pages 25 - 34.

ø Prograaøing In an Interactive Environaentø the LISP Experience by Erik Sandewall. It appeared in March,

1978, in the ACM Computing Surveys, Volume 10:1, pages 35 - 71.

Each of these articles was reprinted in the book Interactive Prog raøøl ng Envl ronaents by David R. Barstow, Howard E.

0THER REFERENCES THAT WILL BE USEFuL T0 You 441
I

OTHER REFERENCES THAT WILL BE USEFUL TO YOU

Shrobe, and Erik Sandewail. This book was published in 1984 by McGraw Hill, NY. The first article can be found on pages 19 - 30, the second on pages 83 - 96, and the third on pages 31 80.

J:'

ill OTHER REFERE~E5 THAT WILL 55 usEFuL TO You

I

----- End Forwarded Messages -----
—*End of message*—