CROCK

By: Kelly Roach (Roach.pa @ Xerox.ARPA)

CROCK (and .DCOM) contain functions for creating and manipulating an analog face clock. It requires that processes be running.

(CROCK REGION)

First invocation creates a clock window, CROCKWINDOW, occupying REGION with style CROCK.DEFAULT.STYLE. If REGION is left NIL, a region will be prompted for. Subsequent invocations use CROCKWINDOW. Only one clock window may exist at any given time. The clock is updated once a minute.

The clock's style is maintained as a property list and can be found by (WINDOWPROP CROCKWINDOW 'STYLE). There are four independent boolean properties which the user may control: HANDS (the hands of the clock), TIMES (time digits printed where the hands end), RINGS (rings on the clock face), and NUMBERS (12 numbers around the outside of the clock face). The style first used will be CROCK.DEFAULT.STYLE (bound to '(HANDS T TIMES NIL RINGS NIL NUMBERS T) when <LISPUSERS>CROCK is first loaded).

Simplest way to call from init file or other function:

(SETQ CROCK.DEFAULT.STYLE <style>)

(CROCK <region>) You supply <style> and <region>.

Left Button

Requests immediate update (moving hands) of the clock. (Of course, it may take a while for the process scheduler to get to it.)

Middle Button

Presents a menu of commands for modifying the clock's style. Menu item SHOW.STYLE prints the clock style. SETTIME requires network time service.

Right Button

RESHAPEing the CROCKWINDOW causes the clock to change its size to fit the new window region. CLOSEing the CROCKWINDOW deletes the clock process.

DATEFNS

DATEMENU(MONTH YEAR)

Puts up a menu that looks like a calendar and returns a string of the day the user selected. The value returned is of the form "dd-mon-yy 00:00:00" which is an acceptable input to **IDATE**. **MONTH** should be the first 3 letters of the month (e.g. **JUL, DEC**). **YEAR** should be the last 2 digits of the year (e.g. 83). The initial menu will be of the month **MONTH** in the year **YEAR** defaulting to the current month and year. The calendar has the preceding and following months in the upper corners; selecting one of these changes the menu month to the selected month.

DEDITK

By: H. Thompson (H. Thompson.pa @ Xerox.ARPA)

DEDITK allows you to perform the following combinations of operations in DEDIT in a single buttoning:

AFTER; BO
BEFORE; BO
BI; DELETE
BI; REPLACE
REPLACE; BO

It accomplishes these operations by changing the menu to include sensitive peripheries for the relevant commands. Load the file and call (DEDITK) to start it up – the next call to DEDIT will exhibit the new menu.

Non-feature: Undoing these operations, or their uncompounded versions, works correctly but displays BITMAP # xxx UNDONE in the prompt window.

DIRECTORYTOOLS

By: Jim Wogulis (Wogulis.pasa@Xerox)
Greg Wexler (Wexler.pasa@Xerox)

DirectoryTools contains one user function: GRAPH.DIRECTORY which is used for graphing the subdirectory structure under a file pattern. e.g:

(GRAPH.DIRECTORY *FILEGROUP WINDOW CASE.SENSITIVE?*)

FILEGROUP is of the form DIRECTORY uses (e.g. {DSK}<LISPFILES>*)

WINDOW if specified is the window grapher uses to display the graph.

CASE.SENSITIVE? if non-NIL will preserve case, otherwise the nodes will all be in upper case.

This will run a process to build the graph, freeing up the TTY. The time it takes to produce the graph is proportional to the number of files on the directory. GRAPH.DIRECTORY returns the window the graph will be in.

The resulting GRAPHER window is then active.

Left button a node:

This will pop up a menu with two items:

FileBrowser - will create a FILEBROWSER window on the selected subdirectory.

List Files - will create a window and list all the files under the selected subdirectory (use this if for a quick look at the subdirectory if you don't want to wait for FILEBROSWER). The listing can be aborted with ctrl-E.

You can also shift-select the node and have it type in the full name of the selected subdirectory into the current TTY .

Middle button node:

This will type in the full name of the selected subdirectory into the current TTY .

Problems:

Currently this will work only for filedevices that use > to distinguish between subdirectories. So it will work for Floppy, Dlion disk, and NS fileservers , and not for Unix or VMS file servers.

DLIONFNKEYS

By: Greg Nuyens

DLIONFNKEYS provides a set of "logical" keys corresponding to the Dandelion (1108) function keys. This allows Dorado and Dolphin users to take advantage of interfaces designed for the 1108 keyboard. Calling (BUILDFNKEYS) builds a window with 8 keys like the one below:

These keys can be "pressed" by bugging the mouse (left or middle mouse button) inside the key's image. The effect of pressing one of these keys is the same as pressing the physical key on the 1108. (for instance, to change the effect of one of the "logical" keys, call the standard Interlisp-D system function KEYACTION with the name of the key.)

The right hand mouse button gets a menu of window commands.

DLIONFNKEYS is implemented by use of the lispusers package KEYOBJ.

EDITBACKGROUND

By: C. D. Lane

It is possible on Xerox 1108s in the Carol version of Interlisp-D and later to change the background border shade (normally black on 1100's, white on 1132's and the same as the background shade on 1108's). The Interlisp-D function to do this is (CHANGEBACKGROUNDSHADE shade) and though it looks analogous to (CHANGEBACKGROUND shade), the shade argument is interpreted differently.

A normal (black & white) shade consists of 16 pixels (see EDITSHADE in the Interlisp manual). The border shade does also but it covers twice the area of a normal shade. Whereas the normal shade has 4 x 4 pixels, the border shade has 2 x 8 pixels and the pixels in the two rows are twice as tall as normal shade pixels:

```
Background          Background Border
+--+--+--+--+       +--+--+--+--+--+--+--+--+
|15|14|13|12|       | | | | | | | | |
 |11|10| 9| 8|      |15|14|13|12|11|10| 9| 8|
| 7| 6| 5| 4|       | | | | | | | | |
| 3| 2| 1| 0|       | 7| 6| 5| 4| 3| 2| 1| 0|
+--+--+--+--+       +--+--+--+--+--+--+--+--+
```

Thus WHITESHADE and BLACKSHADE look the same for both as does the FUGUE/CAROL background shade (38450 = 2↑15 + 2↑11 + 2↑5 + 2↑1) since it chosen to look the same in both:

```
Background          Background Border
+--+--+--+--+       +--+--+--+--+--+--+--+--+
|XX|14|13|12|       |XX| | |XX| | | |
|XX|10| 9| 8|       |XX|14|13|12|XX|10| 9| 8|
| 7| 6|XX| 4|       | | |XX| | | |XX| |
| 3| 2|XX| 0|       | 7| 6|XX| 4| 3| 2|XX| 0|
+--+--+--+--+       +--+--+--+--+--+--+--+--+
```

However, arbitrary shades will not appear the same on both. Vertical lines are preserved but horizontal lines are not. The function EDITBACKGROUND in EDITBG.DCOM lets you edit both an normal shade and a border shade and see how they combine.

Calling the function (EDITBACKGROUND) brings up the edit window which has two texture editors in the bottom half. The background texture editor is on the left and the border texture editor is on the right. The top half of the window echos the textures, the background within the border texture as it would be on the screen. Buttoning the small box in the center of the window will change the background and border textures on the screen to those in the window.

EDITFONT

By: Kelly Roach (Roach.pa @ Xerox.ARPA)

EDITFONT gives the user functions for creating and editing DISPLAY fonts which can be read and written as STRIKE font files. The following functions are provided to the user:

(1) (EDITFONT FONT). FONT may be any DISPLAY FONTDESCRIPTOR datatype. The user will be presented with a matrix of 257 character bitmaps. The 257th char is the dummy char, used as a default for unspecified chars. Buttoning a character bitmap with the LEFT mouse button will call EDITBM on that character bitmap. Buttoning a character bitmap with MIDDLE mouse button pops up a menu that allows the user to STOP (abort EDITFONT), OK (save EDITFONT's work into FONT), or DUMMY (make selected character an unspecified char).

(2) (BLANKFONTCREATE FAMILY SIZE FACE ROTATION DEVICE FIRSTCHAR LASTCHAR ASCENT DESCENT WIDTH). This function will create a blank font for the user. Charcodes not between FIRSTCHAR and LASTCHAR are left unspecified. WIDTH may be a number, causing WIDTH to be the width of every character; or WIDTH may be a list of

(IPLUS LASTCHAR (IMINUS FIRSTCHAR) 2)

numbers, determining the width for each specified char plus the dummy char. Another good way to get a fresh font is to do a COPYALL of some already existing fontdescriptor (e.g., (COPYALL (FONTCREATE 'GACHA 10))). Use Interlisp SETFONTDESCRIPTOR function to make your fonts known to Interlisp FONTCREATE.

(3) (READSTRIKEFONTFILE FAMILY SIZE FACE FILE). Reads STRIKE font FILE, returning a fontdescriptor.

(4) (WRITESTRIKEFONTFILE FONT FILE). Writes FONT out to STRIKE file FILE. Can be read back in with READSTRIKEFONTFILE. You can put your STRIKE font files on FONTCREATE search path by adding your font directory to Interlisp global FONTDIRECTORIES.

EDITHIST

By: Dave Dyer (D Dyer @ ISIB.ARPA)

EDITHIST is an extension to Interlisp's file package that permanently preserves the history of new versions of files.

EDITHIST should be regarded as an extension to the existing mechanism for recording a "who-edited-last" comment within the body of each function.

Every time a file is remade, a new entry is made in the edit history for the file consisting of a list containing the following information: (DATE AUTHOR FILE (CHANGES) (COMMENTS))

where:

DATE is the value of (DATE)

AUTHOR is the value of (OR INITIALS USERID)

FILE is the full file name of the output

CHANGES is the same list of changed items as is on the filedates property.

COMMENTS is acquired (optionally) by ASKUSER during the makefile

How To Use It

Include (LOAD '<DDYER>EDITHIST.COM 'SYSLOAD) in your INIT.LISP

The default settings of the control parameters for EDITHIST will:

- Ask you to type comments each time a changed file is made with MAKEFILE
- Automatically "capture" files that do not already have an edit history
- Maintain an edit history no more that 30 entries long.

You may or may not wish to change the default setting of two parameters: ASKEDITHIST to control the use of ASKUSER, and LIMITEDITHIST to control the pruning of the edit histories.

ASKUSER

Normally, ASKUSER is invoked each time the file is made and invites you to type some descriptive comments about the edit you just made.

Controls on ASKUSER are implemented by the value of ASKEDITHIST and by several options understood by MAKEFILE.

Permanent settings, controlled by the value of ASKEDITHIST

NIL — do not ask, do not extend the edithistory

NOCOMMENT — do not ask, extend the edithistory with no comments list

COMMENT — ask for comments, extend edithistory

T — add EDITHISTORY to files that do not have one, ask for comments and extend the edit history. This is implemented by adding (INITEDITHIST) to MAKEFILEFORMS. This is the default setting.

Temporary overrides, specified by OPTIONS to MAKEFILE. This is the second argument to MAKEFILE or the first argument to MAKEFILES.

ASK or COMMENT temporarily resets ASKEDITHIST to T, so you will be asked for comments this one time only.

NOASK or NOCOMMENT temporarily resets ASKEDITHIST to NOCOMMENT, so you will not be asked for comments this one time only.

When ASKUSER prompts you with: comments: you get to type one LINE of forms. so either type your comments with no carriage returns, or enclose them in parentheses.

Pruning

EDITHIST entries do not accumulate without bound. When a threshold number of entries have been made (see LIMITEDITHIST below) a pruning process is invoked that tries to reduce the size of the list while preserving as much information as possible. The approximate pruning algorithm is:

1. Preserve the first N entries
2. Among the rest, merge entries that have no comments and that have the same author
3. If still above threshold, merge any entries that have no comments, ignoring authors.
4. If still above threshold, merge all the oldest entries to get below threshold.

The entries pruned by this procedure are merged with the adjacent entries, in which case the FILE AUTHOR and DATE properties become a cons cell specifying a range, the CHANGES list becomes the union of all changes, and the COMMENTS list becomes the concatenation of all comments.

Whenever the edit history is pruned, a message is printed giving the new length. Once phase 4 pruning is in effect, the history will be shortened every time the file is made.

At some point, you will undoubtedly want to manually edit the edit history, to remove inconsequential entries. See EDITDEF above.

Controls on pruning of the edit history are implemented by LIMITEDITHIST, which should be (CONS #softlimit #hardlimit). The default value is (10 . 30) which will keep the 10 most recent entries inviolable, and will invoke pruning when the list grows to 30 entries.

Internals

The basic entries for each edit history list are kept in an ALIST keyed by the name of the edit history: eg. the name of the file. Each edit history is ordered chronologically, oldest first. The root of the alist is EDITHISTALIST, but to examine/modify edit histories, use (EDITDEF 'NAME 'EDITHIST) and (SHOWDEF 'NAME 'EDITHIST), where NAME defaults to the NAME.EXT of the file.

The edit histories are declared DONTCOPY, so will not appear in compiled files. The history will be recovered from the original source file if it is remade without being fully loaded, as would occur if you edited a function in the compiled file causing its EXPR definition to be loaded from the source file.

If a single (EDITHIST --) command on the file coms contains more than one edithistory, all but the first are considered dormant, and not added to. at MAKEFILE time. This provides a means of archiving the edit history of a file that has been created by merging several other files.

A Sample Edit History

```
(EDIT.LISP
    (" 8-Jul-80 18:34:41"
    DD: <DDYER>EDITHIST.LISP.26
    (EDITHISTCOMS EDITHIST.LISP))
    (" 8-Jul-80 21:16:01"
    DD: <DDYER>EDITHIST.LISP.27
    (MAKEDITHIST)
    (added output file to the history list entry))
    (" 8-Jul-80 21:25:48"
    DD: <DDYER>EDITHIST.LISP.28
    (EDITHIST.LISP)
    (cleaned up edit history list))))
```

EDITMACROS

Some useful (tty-based) editor macros; includes COPY and SWITCH, with syntax like basic MOVE command, and IFY which finds the "next" COND and converts it to an IF.

EMACS

By: Keily Roach (Roach.pa @ Xerox.ARPA)

A very small subset of PDP-10 EMACS (text editor developed at MIT) built on top of TEDIT. The package is currently in a developmental stage, but is fairly useful as is.

Users should familiarize themselves with TEDIT and TEDIT documentation first and are warned that EMACS undoubtedly "hacks" TEDIT to a certain extent. Interlisp EMACS is not a complete implementation of PDP-10 EMACS and there is a wish list of about a dozen items that still need to be implemented to make Interlisp EMACS completely attractive compared to PDP-10 EMACS. But TEDIT features such as the mouse selection, multiple windows, and communication with Lisp, help to make up for some of the shortcomings.

To use, call
(EMACS TEXT WINDOW DONTSPAWN PROPS)
where arguments TEXT WINDOW, DONTSPAWN, and PROPS are used just as they are with the TEDIT function TEDIT. Examples:

```
(EMACS 'MYFILE)
(EMACS "THIS IS A STRING" (CREATEW NIL "A WINDOW") )
```

It is possible to have multiple EMACS windows.

The following commands are supported:

↑A	Go to beginning of line
↑B	Go back a char
↑D	Delete a char
↑E	Go to end of line
↑F	Go forward a char
↑K	Kill line
↑L	Redisplay window
↑N	Go down a line
↑P	Go up a line
↑S	Forward search
↑T	Transpose chars
↑V	Go forward a windowfull

↑X↑V	Visit file
↑X↑W	Write to file
↑X↑Z	READ & EVAL expression in EXEC
↑Z↑F	Forward an expression
DEL	Delete back a char
#↑A	Go to beginning of definition
#↑E	Go to end of definition
#↑K	Kill expression
#>	Go to end of file
#<	Go to beginning of file
#B	Go back a word
#D	Delete a word
#E	DEDIT expression
#F	Go forward a word
#G	PRETTYPRINT expression
#S	Steal top selection from DEDIT
#V	Go down a windowfull
#↑	Join lines
#DEL	Delete back a word

The following items are still in the wish list category. The authors welcome implementations of any of these from other User experts.

<TAB>	(a la LISP mode)
↑U	(arguments)
↑Y	(remember deleted material)
$X	Replace String
$S	Query Replace
↑R	(reverse searching, faster searching)
↑Z↑B	(nontrivial backward commands)
$X	Auto Fill
$X	Auto Save
$X	Occur
$↑T, ↑X↑T	(nontrivial transposes)

EMACSUSER

By: Kelly Roach (Roach.pa @ Xerox.ARPA)

A few people besides myself have been brought up in the tradition of EMACSing your code rather than DEDITing it. The usual practice is to escape to EMACS, edit your source file a bit, then escape to LISP with the editted code loaded in. This package defines a few functions and CLISP words to support this kind of activity.

(1) FUNCTION DEFINITION. Functions can be defined with clisp words DEFEXPR and DEFFEXPR:

<def> => (DEFEXPR <protocall> . <body>) !
(DEFFEXPR <protocall> . <body>)
<protocall> => (<name> . <wrappers>)
<wrapper> => <formal> !
(OPTIONAL <formal>) !
(OPTIONAL <formal> <default>) !
(REST <formal>) !
(REST <formal> <default>)

The <body> in a <def> is a list of expressions that PROGN might take. The <wrappers> should be a list of the obligatory formals (if any), followed by the optional formals (if any), followed by the rest formal (if there is one).

Functions defined using DEFEXPR expand into LAMBDA forms and they're actuals are EVALuated before being called. Functions defined using DEFFEXPR expand into NLAMBDA forms and they're actuals are not EVALuated before being called. Examples:

```
(DEFEXPR (FOO X (OPTIONAL Y 3) (REST Z 'MEEF)) (LIST X Y Z))
(FOO) => (NIL 3 MEEF)
(FOO 4 5) => (4 5 MEEF)
(FOO 4 5 6 7) => (4 5 (6 7))

(DEFFEXPR (BAR X (REST Y)) (LIST 'TIMES X (CONS 'PLUS Y)))
(BAR 1 2 3) => (TIMES 1 (PLUS 2 3))
(BAR A B C) => (TIMES A (PLUS B C))
```

(2) ARITHMETIC. Shorter function names for arithmetic functions are supplied.

	FIXP	FLOATP
PLUS	+	+$
TIMES	x	x$
DIFFERENCE	-	-$
MINUS	0-	0-$
QUOTIENT	/	/$
REMAINDER	\	\$
ADD1	1+	1+$
SUB1	1-	1-$
GTHAN	>	>$
GEQUAL	>=	>=$
NEQUAL	<>	<>$
EQUAL	=	=$
LTHAN	<	<$
LEQUAL	<=	<=$

(3) BQUOTE & SETF. This package also LOADs <LISPUSERS>BQUOTE & SETF.

(4) LOAD, EFILE & ECOMPL. You can LOAD an EMACS file just like you would LOAD a filepackage file. But to keep LOAD happy, you have to have the atom STOP at the end of your file. You can convert an EMACS source file into a typical INTERLISP filepackage source file and compile it. To create a filepackage file from an EMACS file, use

(EFILE FILE TOFILE)

The function

(ECOMPL FILE)

does an EFILE followed by a TCOMPL.

EXEC

This small package allows the user to create extra EXEC windows in which to do EVALQTing. A new EXEC window may be created in two ways. The user may either do (EXEC) or button the EXEC menu item added to the background menu. Typing (EXEC.QUIT) into an EXEC window will close the window (function EXEC.QUIT also takes a WINDOW argument and can be used as a CLOSEFN window property).

EXECFNS

By: Jeffrey S. Shulman (Shulman @ Rutgers.ARPA)

The purpose of this package is to emulate TOPS-20 logical names.. The full DEF macro is:

DEF logical-name: [*definition*]

Where logical-name is any logical name you wish and the optional field definition is the definition. If logical-name: is already defined then definition will replace the old definition. If definition is not present then logical-name: will be removed from the definition list. All current logical definitions are kept in a list format on the global variable *LOGICAL.DEFINITIONS*.

The lispxmacro DEF? can be used to see what logical definitions are currently defined. The format of DEF? is:

DEF? [logical-name:]

where logical-name: is either a logical name or a '*' (leaving logical-name: absent is the same as '*'). You will be told what logical-name: is defined as or given a list of all the currently defined logical names is you give * (or nothing).

The lispxmacros TY, CONN and KD (defined below) have been (re)defined to expand logical names to their full file definition. In addition the functions INFILE, OUTFILE, INFILEP, OUTFILEP and COPYFILE (see further comments on COPYFILE below) have also been modified/advised to expand logical names.

The function that implements DEF is:

(DEFINE.LOGICAL.DEVICE logical-name definition)
DEF? is implemented by:

(SHOW.LOGICAL.DEFINITIONS logical-name)
The function

(LOGICAL? filename)

returns the full file name of filename with any logical names fully expanded.

The lispxmacro KD and function KDELETE implement the TOPS-20 KDELETE command. They both take as arguments a file name (with wildcards) and will delete all but the most recent version of the matching files.

The function DEFAULTFILE will fill in possible missing file name and extension in its first argument from its second. The function COPYFILE has been implemented to use this function to default to TOFILE name.

Both FROMFILE and TOFILE will expand logical names. The function COPYFILE has been modified to preserve the file type between file tranfers. When tranferring from file server to Dolphin, it uses the GETFILEINFO function to get the file type. When tranferring from the Dolphin, it accepts an optional third argument being either TEXT or BINARY to specify the file type. It defaults to TEXT if not given.

FINGER

By: Greg Nuyens (Nuyens.pa @ Xerox.ARPA)

Description:

Finger is a facility for determining and displaying information about other users running Interlisp-D. It displays the user's name, the Etherhostname (or the octal net address when no nameserver is available) and the user's idle time (time since last keystroke or mouseaction). Only other users who have the finger server loaded will be displayed. Users can specify the net radius to query, a list specifying only which users they want displayed, or similarly, only which hosts are to be displayed.

Use:

Loading the file begins the finger server (which responds to queries). To display finger information, the following top-level function is provided:

(FINGER **who host hops icon?**)

who is an optional list of usernames specifying which people are to be displayed if a response is received. **who** defaults to FINGER.CROWD, initially NIL meaning display all response.

host is an optional list of etherhostnames analogous to **who**. Specifying both **who** and **host** denotes union. **host** defaults to NIL, denoting all hosts.

hops specifies the net radius to query. 0 specifies only nets to which you are directly connected. **hops** defaults to FINGER.NET.HOPS, initially 2.

icon? specifies whether initial display should be the finger icon or a display window. **icon?** defaults to NIL meaning display. {typically, in an init file the call would be (FINGER NIL NIL NIL T)}.

The display window is updated each time the user bugs the display window with left or middle mouse button, and when most window operations are performed on the display window (shape, repaint, expand from icon, etc.). Right button retains the standard window menu.

Options:

The following are user specifiable variables affecting the operation of Finger.

FINGER.ICON.POSITION a position indicating the original position for the icon. Initially (900,500).

FINGER.DISPLAY.POSITION a position indicating the original position for the display window. Initially (650,325).

FINGER.DISPLAY.HEIGHT height of the display window. Initially 140. The display width is correct for the display format and need not be changed.

FINGER.TIMEOUT milliseconds to wait for the last response packet. Initially 1500.

FINGER.NET.HOPS net radius to be queried.

FINGER.CROWD list of potential users to be displayed (discussed above).

FINGER.INFINITY.MINUTES number of minutes to be considered infinite idle time. Initially 90.

Additional functions of interest to the user are:

(END.FINGER) which kills the finger server process, closes the sockets, closes the windows, etc.

(FINGER.SERVER) will start a finger server process.

FONTMENU

By: Ken Feuerman (Feuerman.pa @ Xerox.ARPA)

An Interlisp-D Package for Selecting Fonts from Menus

FONTMENU provides a user-friendly means for presenting fonts and allowing a user to select one. It takes advantage of tree-menus with SUBITEMS to provide a hierarchical presentation of the fonts for the user. The fonts that are listed on the tree menu are displayed by the words which describe the font (i.e., **HELVETICA 10BOLDITALIC**), and displayed in the font that these words describe. Thus the user has a picture of what the font looks like before actually selecting it.

The menu itself is a tree menu. The first menu that appears, labelled "Fonts," provides a list of families of fonts to choose from, such as **GACHA ,TIMESROMAN** ,etc. Pulling off to the right from one of these families will display another menu of the family in various sizes. Pulling off to the right of one of these sizes will display a menu with family, size, and face specified. A selection may be made at any time, without having to pull off to the right to the last level of menus. The menu will return a **FONTDESCRIPTOR** of the selected font.

(FONTMENU.CREATE *FONTLIST DEFAULTSIZE DEFAULTFACE*) [Function]

Returns the font menu. *FONTLIST* is the variable that describes the fonts to be presented on the menu. It is a list of family specifications.

A family specification is a list, the **CAR** of which is a litatom referring to the name of the family of the font (**GACHA ,HELVETICA** ,etc.) The **CDR** of the family specification is a list of size specifications and/or default size or face specifications.

A size specification is a list again. The **CAR** of a size specification is the size number (10 point, 12 point, etc.), and the **CDR** is a list of faces and/or default face specification. A face is any form acceptable to **FONTCREATE (STANDARD, BOLDITALIC, (MEDIUM ITALIC REGULAR)**, etc.).

DEFAULTSIZE determines how the family names will appear in the menu of family names only (first level of the tree). *DEFAULTFACE* determines the face in which the family names and sizes will appear in the first two levels of the font menu. If a selection is made from one of these two levels (without pulling all of the way off to the right to fully specify a font), a fontdescriptor is

returned using the values of *DEFAULTSIZE* and *DEFAULTFACE* (i.e., exactly as it appears in the menu). If not given, 10 is used for *DEFAULTSIZE* and **STANDARD** = (MEDIUM REGULAR REGULAR) is used for *DEFAULTFACE*

Furthermore, the default size and/or face for each font at each step in the tree menu can be specified by including a default face or default size specification. These simply take the form **(DEFAULTSIZE . 24)** and **(DEFAULTFACE . BOLD)**, and may be included at any non-conflicting step in the *FONTLIST* (by "non-conflicting," it is meant that it doesn't make sense to include something like **(DEFAULTSIZE . 24)** inside of a size specification, since the size specification already has its size defined).

NOTE: Since each of the fonts specified in *FONTLIST* must be created (using **FONTCREATE**), this function could take a very long time to execute.

FONTLIST [Variable]

A list suitable as a *FONTLIST* variable to FONTMENU.CREATE . Its value is initially

```
( (CLASSIC      ( 8 STANDARD ITALIC BOLD)
                (10 STANDARD ITALIC BOLD)
                (12 STANDARD ITALIC BOLD)
                (14 STANDARD ITALIC BOLD)
 (CREAM         (10 STANDARD ITALIC BOLD BOLDITALIC)
                (12 STANDARD ITALIC BOLD BOLDITALIC) )
 (GACHA         ( 8 STANDARD ITALIC BOLD BOLDITALIC)
                (10 STANDARD ITALIC BOLD BOLDITALIC)
                (12 STANDARD ITALIC BOLD BOLDITALIC) )
 (HELVETICA     ( 5 STANDARD ITALIC BOLD BOLDITALIC)
                ( 7 STANDARD ITALIC BOLD BOLDITALIC)
                ( 8 STANDARD ITALIC BOLD BOLDITALIC)
                ( 9 STANDARD ITALIC BOLD BOLDITALIC)
                (10 STANDARD ITALIC BOLD BOLDITALIC)
                (11 STANDARD ITALIC BOLD BOLDITALIC)
                (12 STANDARD ITALIC BOLD BOLDITALIC)
                (13 STANDARD ITALIC BOLD BOLDITALIC)
                (14 STANDARD ITALIC BOLD BOLDITALIC)
                (16 STANDARD ITALIC BOLD BOLDITALIC)
                (18 STANDARD ITALIC BOLD BOLDITALIC)
                (36 STANDARD ITALIC BOLD BOLDITALIC) )
```

```
(HELVETICAD    (DEFAULTSIZE . 24)
               (DEFAULTFACE . STANDARD)
               (24 STANDARD ITALIC BOLD BOLDITALIC)
(OLDENGLISH    (10 STANDARD ITALIC BOLD BOLDITALIC)
               (18 STANDARD ITALIC BOLD BOLDITALIC)
(TIMESROMAN    ( 4 STANDARD ITALIC BOLD BOLDITALIC)
               ( 6 STANDARD ITALIC BOLD BOLDITALIC)
               ( 8 STANDARD ITALIC BOLD BOLDITALIC)
               ( 9 STANDARD ITALIC BOLD BOLDITALIC)
               (10 STANDARD ITALIC BOLD BOLDITALIC)
               (11 STANDARD ITALIC BOLD BOLDITALIC)
               (12 STANDARD ITALIC BOLD BOLDITALIC)
               (13 STANDARD ITALIC BOLD BOLDITALIC)
               (14 STANDARD ITALIC BOLD BOLDITALIC)
               (16 STANDARD ITALIC BOLD BOLDITALIC)
               (18 STANDARD ITALIC BOLD BOLDITALIC)
               (36 STANDARD ITALIC BOLD BOLDITALIC) )
(TIMESROMAND   (DEFAULTSIZE . 24)
               (24 (DEFAULTFACE . STANDARD)
                   STANDARD ITALIC BOLD BOLDITALIC)
               (30 (DEFAULTFACE . ITALIC)
                   STANDARD ITALIC BOLD BOLDITALIC)
               (36 (DEFAULTFACE . BOLD)
                   STANDARD ITALIC BOLD BOLDITALIC) ) )
```

and is intended to represent most of the possible fonts. It is recommended that one COPY this variable to a new variable, and then edit it to represent the desired list of fonts.

FONTN

By: Ron Kaplan (Kaplan.pa @ Xerox.ARPA)

FONTN

This file contains font definitions for fonts FONT1, FONT2, etc., which are utilized by the printout package for commands of the form .FONT 1, .FONT 2, etc.

GRAPHCALLS

By: Christopher Lane (Lane @ SUMEX-AIM)

GRAPHCALLS is an extended graphical interface to the Interlisp CALLS function. It is to CALLS what BROWSER is to SHOW PATHS in MASTERSCOPE. It allows fast graphing of the calling hierarchy of both interpreted and compiled code, whether or not the source is available (see the CALLS function in the Interlisp manual), allowing examination of both user and system functions. The functions do not have to be analyzed by MASTERSCOPE first.

Additionally, buttoning a function on the graph brings up a menu of operations that can be done with the function, such as editing, inspecting, further graphing etc.

(GRAPHCALLS function filter depth flags format)

Graphs the calling hierarchy of function. Terminal nodes on the graph (those who call no other functions or are undefined) are printed in a bold version of the graph's font indicating that they cannot be graphed further.

The filter argument is a function to apply to the functions when building the graph to test their eligibility to appear on the graph. The filter can be any defined function; the default is not to filter. Interesting filters are:

WHEREIS	Limits the tree to only functions the user as has loaded and prunes out Interlisp functions and SYSLOADed files. Quite useful.
FGETD	Limits the tree to only functions that are actually defined. Thus if you are perusing the tree for BITBLT and don't have and are not interested in the color code, FGETD will remove all of the undefined color bitmap functions.
EXPRP	Limits the tree to interpreted functions. Useful for graphing functions in the development stage.
CCODEP	Limits the tree to compiled functions.
NO\	Keeps low level functions starting with \ (i.e. \OUTDATE) off of the graph. Useful for getting an overview of system functions and when advising system functions (as \'ed functions probably should not be advised).

The calling hierarchy is graphed to depth grandchildren (defaults to 1). A depth of 2 would pick up greatgrandchildren etc.

The flags argument can be one or more of:

INVERT	Sets up a dynamic graph to visually track a running program 2 (see Dynamic Graphing below).
COUNT	Sets up a dynamic graph to count function calls in running program (see Dynamic Graphing below).
EDIT	Passes T to SHOWGRAPH's alloweditflg, allowing editing of the graph from a right button menu.
TOP	Passes T to SHOWGRAPH's topjustiflyflg.

The format argument is passed on to LAYOUTGRAPH in GRAPHER and can be any format specification (eg. LATTICE, VERTICAL, FAST, REVERSE etc.) In the forest format multiple instances of a function appear on the graph after every calling function and a boxed node indicates the function appears elsewhere on the graph, possibly graphed further. In the lattice format each function gets placed on the graph only once (particularly useful for for the Dynamic Graphing described below), and boxed nodes indicate recursive functions calls.

Graph Menus

The menu that pops up when you left button a function on the graph contains the following items (NOTE: The subitems are implemented as menu SUBITEMS and are also available by pressing the middle button over the item.):

?=	Print the arguments to the function, if available.
FNTYP	Print the function's FNTYP.
WHERE	Do a WHEREIS on the function.
EDIT	Calls the editor on the function if available for editing.
TYPEIN	BKSYSBUF's the name of the function into the typein buffer.
BREAK	Applies BREAK to the function. Its subitems are

BREAKIN Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.

UNBREAKIN Undoes BREAKIN.

UNBREAK Applies UNBREAK to the function.

TRACE Applies TRACE to the function.

TRACEIN Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.

CCODE Calls INSPECTCODE on the function if it is compiled code.

GRAPH Calls GRAPHCALLS to make a new graph starting with function, inherits the original graph's filter, flags and format.

FRAME Inspect the local, free and global variables of the function. These are the last three lists of the CALLS function placed into INSPECT windows. Its subitems are

>FRAME Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.

<FRAME Like >FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.

Buttoning the graph outside a node give you a menu with these options:

HardCopy Does a HARDCOPYW on the window with the graph.

UNBREAK Does an (UNBREAK), unbreaking all broken functions.

RESET Resets the counters for the COUNT option and redisplays graph.

Dynamic Graphing

When the INVERT or COUNT flags are given, GRAPHCALLS will advise all of the functions on the graph (in the context of their parent) to invert their corresponding node on the graph (as well as delay some to allow it to be seen) and/or follow each function name by a count of the number of times it has been executed. In invert mode, a node remains inverted as long as control is inside its corresponding function and it returns to normal when the function is exited. The lattice format is best when using the invert feature. Closing the graph window UNADVISEs the functions on the graph.

A simple example of this is (GRAPHCALLS 'DATE NIL NIL 'INVERT) and then evaluate (DATE).

GRAPHCALLS will not graph or advise any function in the list NOADVISEFNS when an advise flag is used. Functions which are unsafe to advise should be added to this list.

CAVEAT PROGRAMMER! This feature must be used with caution. As a rule, one should not do this to system (Interlisp) functions, but only one's own, use WHEREIS as a filter for this. Advising system code indiscriminately will probably crash the machine unrecoverablely.

You can, at some risk, interactively BREAK and EDIT etc. a function on the graph while the code is executing. Also, creating subgraphs of advised graphs will show the generated advice functions not the original functions called, as will creating new graphs of functions in advised graphs. You can create advised graphs of functions already graphed normally on the screen.

Command Window

The function (GRAPHCALLSW) puts up a command window with menus that will interactively set up calls to GRAPHCALLS. The menus let you set the Invert, Count and Edit flags as well as select from common filters and formats as well as set the depth of the graph. You can also change the amount of delay used in the advised functions when doing Dynamic Graphing. If you specify an advised graph (Invert or Count) and don't specify a WHEREIS filter, you will be asked to confirm with the mouse for your own protection.

More than one item on the filter and flags menus can be selected at a time. Buttoning a selected item on these menus unselects it.

The command menu contains the following:

Function — Sets the current function which the command graphs by prompting for a name in the prompt window.

Include — Adds files or functions to the list of items to allow on the graph, see the Include/Exclude algorithm below.

Exclude — Adds files of functions to the list of items disallowed on the graph, see the Include/Exclude algorithm below.

Clear — Clears all of the settings on the command window to the defaults. Also clears the Include/Exclude lists.

GRAPH — Graphs the function by calling GRAPHCALLS with the selected options.

Include and Exclude allow fine tuning of the filter function. If the function passes the filter, then the following are tried until one determines whether or not the function will be on the graph

1. If a set of functions has been explicitly excluded, and the function is a member of this set, it will NOT appear on the graph.

2. If a set of functions has been explicitly included, and the function is a member of this set, it WILL appear on the graph.

3. If a set of files has been explicitly excluded, and the function is in one of those files, it will NOT appear on the graph.

4. If a set of files has been explicitly included, and the function is not in one of those files, it will NOT appear on the graph.

5. The function WILL appear on the graph.

When the command window is open, middle buttoning a node on any GRAPHCALLS graph will bring up a menu of commands relating to command window and graphs. The menu contains:

EXCLUDE — Adds the function to the exclude functions list of the command window. This is the only way to exclude system functions which get added to the SYSTEM file exclusion list.

The command window can also be obtained via the BackgroundMenu. Subsequent calls to GRAPHCALLSW (either directly or via the BackgroundMenu item) will reuse the old command window if there is one. If this window is damaged, and redisplay doesn't help, then (GRAPHCALLSW T) will build a new command window from scratch.

Notes:

Function call graphs are constructed using breadth first search but GRAPHER lays out graphs using a depth first approach so that in some cases functions are expanded in a different place on the graph than one would expect.

GRAPHCALLS sysloads GRAPHER if it is not already loaded.

In dynamic graphs, variables caused by advising show up in the frame inspections.

Subgraphs made from graphs created from the command window will inherit the current graph filter, which may be for a completely different context.

The font for the graph (initially GACHA 8) can be changed by setting the global CALLSFONT.

The delay in advised graphs can be changed by setting CALLSDELAY which is reset by the command window when using the delay menu.

HEADLINE

By: D. Austin Henderson, Jr. (A Henderson.pa @ Xerox.ARPA)

HEADLINE contains three functions for manipulating windows which contain headlines ("headline windows").

HEADLINE (PHRASE FONT POSITION ALIGNMENT)

Creates a headline window with **Phrase** printed in font **Font** at position **Position** aligned as per **Alignment**; the window is just large enough to hold the headline. **Phrase** is any Lisp object. **Font** defines a font as per FONTCREATE (eg. (TIMESROMAN 18 BOLD)); if NIL, TimesromanD 36 is used. **Position** is a position giving the reference point for placing the window; if NIL, the user is given a chance to position the window with MOVEW. If **Postion** is given, **Alignment** gives the alignment of the window with respect to **Position** as (xalignment . yalignment) where xalignment is one of LEFT, CENTER, or RIGHT and yalignment is one of BOTTOM, CENTER, or TOP; for convenience, if Position is CENTER then it is taken to mean (CENTER . CENTER), etc.

BILLBOARD (TITLES ALIGNMENT SEPARATION POSITION)

Creates a set of vertically arranged headline windows. **Titles** is a list of (phrase font) sublists where phrase and font are as in Headline. **Alignment** is one of LEFT, CENTER, or RIGHT, indicating how the windows are aligned with each other; defaults to CENTER. **Separation** indicates the spacing between the bottoms of the windows; defaults to 70. **Position** indicates where the top (first) of the windows is to appear; defaults to somewhere near the top center of the screen.

CLOSEHEADLINES ( )

Closes all the active headline windows.

HISTMENU

By: Daniel G. Bobrow (Bobrow.pa @ Xerox.ARPA)

The history menu package provides a simple way to access the history list using a menu. Through this menu one can REDO, UNDO, FIX, and ?? selected items. One can also delete selected items from the menu.

(HistoryMenu N) will create a Menu of the last N history events in a HistoryWindow. HistItemsShown items of these N items will be seen on the menu. The rest may be seen by scrolling the menu. N defaults to HistDefaultSlice.

Selecting an item with LEFT in the window does a REDO on that item. Selecting an item with the MIDDLE button and then releasing the button causes a popup menu to be shown which provides a set of other options: FIX, UNDO, ??, and Delete. Selecting FIX, UNDO, and ?? cause that history operation to be done on the item selected. Look in the prompt window while holding down a button on a selection to see what will happen if you let the button up. Clicking the RIGHT button in the window will bring up the usual window options, plus an UPDATE option which will update the menu to contain all current items.

Delete causes an item to be deleted from the menu (not the real history list). Deleting "unuseful" items allows more "good" items to appear in the same window space. Also, atomic items on the list BadHistoryItems will never appear on the History Menu. BadHistoryItems is initially (EDIT ?= OK T NIL ↑)

Update makes the menu contain all of the current slice of history. The History Menu is NOT automatically updated as you type in. However, after each deletion, the HistoryMenu is updated automatically if UpdateOnDeleteFlg is not NIL. UpdateOnDeleteFlg starts as T.

(HistIcon N histPosition iconPosition) will create a history menu at histPosition, and then shrink it to a special picture icon (which looks like a scroll) located at iconPosition.

The HistoryWindow and HistoryMenu creation are controlled by the following parameters in addition to BadHistoryItems and UpdateOnDeleteFlg.

Name	Initial Value	Explanation
HistDefaultSlice	30	Number of items in HistoryMenu
HistItemsShown	51	Number of items shown in window
HistMenuHeight	15	Height of an item in the menu in screen bits
HistMenuWidth	164	Width of item in Menu in bits
HistWindowWidth	164	Width of HistoryWindow in bits.
HistEventWidth	60	Maximum number of characters that will be shown of the event in the prompt window

KAL

By: Michel Denber (Denber.wbst @ Xerox.ARPA)

KAL is a program which draws kaleidoscopic patterns in its window on your screen. You can also run it in color if you have a color display connected to your machine. KAL runs as a process so you can leave it running while doing other work.

To run it, load the file KAL.DCOM and then type (KAL). It will ask you if you want to run in color. Select your choice and enjoy. To stop KAL, use the middle mouse button to bring up a pop-up menu. Select STOP. The menu also contains choices that let you vary two of the drawing parameters - PERIOD and PERSISTENCE. If you select these items, you will be prompted for new values in the Prompt Window.