KEYOBJ

By: Greg Nuyens

KEYOBJ provides an Interlisp-D imageobject which mimics a key. The default image looks like this:

These keys are pressed by clicking the mouse inside the key's image. The result of pressing a key is determined (just like the physical key) by the Interlisp-D system function KEYACTION. To enter a KEYOBJ into TEdit type ↑o. Inside the window that pops up, call the following function:

(KEYOBJ.CREATE KeyName KeyLabel) [Function]

KeyName is the key that you want the object to behave like. (CENTER in the example above). KeyLabel is an optional label other than the key whose action it mimics. If KeyLabel is a list of two elements, the first is displayed above the second.

KEYOBJ.FONT [Variable]

Determines the font in which the label is created inside the keyobj. Default is Helvetica 1

LABEL

By: Larry Masinter (Masinter.pa @ Xerox.ARPA)

This implements the LABEL construct of Lisp 1.5, permitting inline specification of recursive functions.

(LABEL name args . forms)

Can be used inline in place of an open lambda. The difference is that forms may contain (lexically, not hidden under a macro) forms that look like calls to a function name (e.g. (name a b)). These calls will apply (recursively) the function specified by the LABEL.

Currently this is implemented by binding name as a variable with value the lambda expression (LAMBDA args . forms), except that (name a b) forms are converted to (APPLY* name a b) and (FUNCTION name) forms are converted to just name.

LABEL requires and will load the LAMBDATRAN package.

LINEDEMO

This package contains a couple of random demonstration programs:

(POLYGONS W NOBLOCK TIMER)

Call (POLYGONS) or (POLYGONS window) to perpetually draw polygons in the given window (it (re)uses POLYGONSWINDOW if argument is NIL). To run in the background, you can ADD.PROCESS((POLYGONS (CREATEW]. Controlled somewhat by the global parameters POLYGONMINPTS (minimum number of vertices), POLYGONMAXPTS (maximum number of vertices), POLYGONSTEPS (number of steps between min and max), and delays POLYGONWAIT (time between different polygons) and POLYGONWAIT2 (delay between initial display of beginning and end and the movement phase.)

If NOBLOCK is T, it doesn't block at all (runs after but can't run in background). If TIMER is given, then POLYGONS will stop after TIMER is expired. (Used by the demo system.)

(LINES W)

Similar to POLYGONS in controls, but draws perpetually changing form using line draw. Controlled by single variable LINECNT, which is read at startup as number of lines visible at any one time.

Both POLYGONS and LINES adjust themselves to the size of the window, so you can reshape the window in the middle of the demo.

This file also has a 'fadeto' primitive, for showing one bitmap slowly fading into another in a window.

(FADETO TO FROM WAIT) will modify FROM to look like TO.

LOADFILES

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

This file defines the function LOADFILES to help in loading groups of files.

loadfiles[dir;files;ext;ldflg;printflg]

Loads the files specified by the file argument as modified by the dir and ext specification. files may be a single file name or a list of file names. If a filename includes an explicit directory and extension, then it is simply loaded. Otherwise, dir (if given) is packed on as the directory for each filename without an explicit directory, and ext is packed on as an extension for each file without an extension. ldflg and printflg are the same as the arguments to LOAD. (Note that if ldflg is SYSLOAD, the indicated files will be made part of the "system.")

The argument ext can be an atomic extension (without or without the period), but it provides somewhat more flexibility. It can also be a "preference" list of extensions. In this case, loadfiles will scan through the list of extensions looking for the first one that exists and load that one. Thus (COM NIL) means attempt to find a compiled file, but load the symbolic (with no extension) if the .COM version doesn't exist.

The atomic extension COM is actually handled specially: it is mapped internally into the list (DCOM COM NIL). Loadfiles will print the full-file-name of each file that it loads; it will cause a file-not-found error for each file that it cannot locate. The value of loadfiles is a list of the files actually loaded.

LSET

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

LSET contains functions for operating on sets implemented as lists. A given function like LUNION comes in several flavors.

LUNION takes multiple arguments and uses EQUAL
LUNION2 takes 2 arguments and uses EQUAL
LUNIONQ takes multiple arguments and uses EQUAL
LUNIONQ2 takes 2 arguments and uses EQ

The file currently contains the following functions:

UNION: LUNION, LUNIONQ, LUNION2, LUNIONQ2
INTERSECTION: LINTERSECT, LINTERSECTQ, LINTERSECT2, LINTERSECTQ2
SUBTRACTION: LSUBTRACT, LSUBTRACTQ, LSUBTRACT2, LSUBTRACTQ2

Other useful set functions that are included are:
(LUNIFY E L) Returns LUNION of (LIST E) and L.
(LUNIFYQ E L) Returns LUNIONQ of (LIST E) and L.

(LSETIFY L) Returns L with all EQUAL duplicates deleted.
(LSETIFYQ L) Returns L with all EQ duplicates deleted.

(LSUBSET X Y) Returns T iff X is a subset of Y.
(LSUBSETQ X Y) Similar to LSUBSET but uses EQ.

(LEQUAL X Y) Returns T iff X equals Y.
(LEQUALQ X Y) Returns T iff X equals Y using EQ.

Functions like LUNION and LUNION2, which do tests with EQUAL, assume that the args that are passed to them are in LSETIFY form--that is, there are no EQUAL duplicates within any arg. Functions like LUNIONQ and LUNIONQ2, which do tests with EQ, assume that the args that are passed to them are in LSETIFYQ form--there are no EQ duplicates within any arg. It isn't necessary that the user call LSETIFY or LSETIFYQ, if the user can guarantee that the args to LUNION, etc., do not have EQUAL or EQ duplicates, respectively.

MAGNIFYW

By: C. D. Lane (Lane @ SUMEX-AIM)

MAGNIFYW.DCOM is a software magnifying glass for examining the Interlisp-D screen. The program magnifies the bits underneath the cursor, similar to looking at a bitmap in EDITBM. However, MAGNIFYW is more dynamic as the cursor can be moved around at will to examine any part of the bitmap.

(MAGNIFYW WINDOW) changes window's cursor functions so that when the mouse enters the window, the cursor will change to the magnification cursor and the bits underneath the cursor will be echoed 8 times larger in the magnification window. The cursor is restored to normal when the window is exited. The magnification window is created on the initial call to MAGNIFYW. Shrinking the magnification window will turn off magnification until it is expanded again. MAGNIFYW changes the CURSORMOVEDFN, CURSOROUTFN, CURSORINFN, and the CLOSEFN of the window. (UNMAGNIFYW WINDOW) changes these window properties to NIL, turning off magnification of the window.

The function (MAGNIFYSCREEN) changes the cursor to the magnification cursor and will magnify the entire screen to the magnification window until a mouse button is pressed.

MAKEGRAPH

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

MAKEGRAPH is a package which sits on top of Grapher and helps one create graphs depicting a data structure by walking through it. The central idea is that each point in the walk (node in the graph) is characterized by a datum/state pair and motion is defined by a graph specification in the form of state transition function. This function is specified by a collection of state specifications, each of which indicates how to display (label and font) the datum when one is in that state and how to find the datum/state pairs which are the sons of that node. Also the state specification may specify additional roots for the walk. The generation of a branch of the graph ceases when either there are no sons of a node, or an already encountered node is revisited (identical datum and identical state). The package contains a function for creating such graphs and an example of its use: a function which graphs the graph specifications themselves.

Main Functions:

MAKE.GRAPH (WINDOW TITLE GRAPH.SPECIFICATION ROOTS CONTEXT LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG)

Creates a MAKEGRAPH window. If WINDOW is NIL, then a new one will be created and the user will be prompted to position it. Otherwise, the graph will be shown in WINDOW. The window will be titled with TITLE, will call LEFTBUTTONFN and MIDDLEBUTTONFN on nodes selected (or NIL if selection is made where no node is positioned), and will be justified as indicated by TOPJUSTIFYFLG (a la Grapher). The button functions are defaulted to MAKE.GRAPH.LEFTBUTTONFN (which scrolls the window so that the selected node is in the middle of the window, or if the left shift key is depressed, prints out information about it) and MAKE.GRAPH.MIDDLEBUTTONFN (which inspects the datum of the node selected, or if the left shift key is depressed, inspects the node itself). The arguments to MAKE.GRAPH are added as properties to the window under their argument names. Selecting in the title invokes the functions which are the values of the window properties TITLE.LEFTBUTTONFN and TITLE.MIDDLEBUTTONFN (not in the calling sequence; set by the user if desired; called with a single argument - WINDOW; defaulted to a function which provides a menu of UPDATE and SHOW.GRAPH.SPEC (see functionality below)). The graph is created according to the GRAPH.SPECIFICATION (see below), starting from ROOTS which are (DATUM . STATE) pairs. CONTEXT is an extra argument which is a passed along to all accessing expressions.

A GRAPH.SPECIFICATION is a property list of STATE.SPECIFICATIONs where the properties are the state names. A STATE.SPECIFICATION is a property list whose properties and values are as follows (in this, EXPR means a LISP form which will be evaluated in an environment in which DATUM is bound to the node's datum, STATE to the node's state, and CONTEXT to context):

LABEL: an expression returning something which will be printed as the label of the node; if no LABEL property is provided, the string "???" will be used.

FONT: an expression returning the font to be used for this node; if no FONT property is provided, the default font for the grapher will be used.

SONS: a form indicating a list of (DATUM . STATE) pairs to be used in generating the sons of this node; the acceptable forms are any of the following:

(data-expression state-expression) where data-expression returns a list of datum's for the son nodes, and state-expression is evaluated in the context of each of these in turn to produce the corresponding state of each.

(LIST (datum-expression state-expression) ... (datum-expression state-expression)) a template of expressions which are evaluated individually to produce a list of sons of the same form, viz. (DATUM . STATE) pairs.

(EVAL expression) the expression returns a list of (DATUM . STATE) pairs of the sons.

(UNION sons-spec ... sons-spec) where each sons-spec is any of these forms (recursively).

(TRACE sons-spec) a device for helping debug graph specifications; the value is the value of sons-specs; the user is given the chance to INSPECT them after they have been generated.

ROOTS: like SONS, except the resulting (DATUM . STATE) pairs are used as possibly additional roots of the graph.

MAKE.GRAPH.CONSTRUCT (GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT)

This is the functional heart of MAKE.GRAPH broken out for those who wish to handle their own interactions with grapher and the window package. It produces a list of graphnodes with labels

and sons as specified by GRAPH.SPECIFICATION (see MAKEGRAPH), starting from INITIAL.ROOTS which are (DATUM . STATE) pairs. CONTEXT is an extra argument which is a passed along to all accessing expressions. Returns the list of graphnodes.

MAKE.GRAPH.FIND.ROOTS (GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT)

Finds the real roots from a set of initial roots, using the same processing as MAKEGRAPH uses. This is helpful when you want to hand a "correct" set of roots of a structure to MAKEGRAPH without having to explore the dependencies within that structure. As with MAKEGRAPH, the data structure is processed according to the GRAPH.SPECIFICATION (see MAKEGRAPH), starting from INITIAL.ROOTS which are (DATUM . STATE) pairs. CONTEXT is an extra argument which is a passed along to all accessing expressions. Returns the real roots as a list of (DATUM . STATE) pairs.

Supporting Functions:

MAKE.GRAPH.UPDATE.WINDOW (WINDOW)

Uses the window properties (which may have been changed) to reinvoke MAKE.GRAPH on the window.

MAKE.GRAPH.SHOW.SPEC (GRAPH.SPECIFICATION)

Uses MAKE.GRAPH to produce a graph of a GRAPH.SPECIFICATION. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.SPEC.SPEC which presents GRAPH.SPECIFICATION (reflectively) as a graph.specification. (GRAPH.SPECIFICATION can serve as a template for other graph.specifications. It is a fairly complex 9-state specification. For a simpler example see below under MAKE.GRAPH.SHOW.LIST.)

MAKE.GRAPH.EXAMPLE ( )

Calls MAKE.GRAPH.SHOW.SPEC on MAKE.GRAPH.SPEC.SPEC.

MAKE.GRAPH.SHOW.LIST (OBJECT)

Uses MAKE.GRAPH to produce a graph of an arbitrary Lisp object. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.LIST.SPEC which presents OBJECT as a tree whose nodes are LISTPs and whose leaves are non-LISTPs. (MAKE.GRAPH.LIST.SPEC is the simple 1-state specification below, included here as an example of a graph.specification.

```
(OBJECT (          DOC (ANY LISP OBJECT)                    some documentation
        LABEL (COND          ((LISTP DATUM) "( )")
                   (T DATUM))
        SONS ((COND          ((LISTP DATUM) DATUM)
                   (T NIL))
              (QUOTE OBJECT))
```

For a more complex example see above under MAKE.GRAPH.SHOW.SPEC.)

MAKE.GRAPH.EXAMPLE.2 ( )

Calls MAKE.GRAPH.SHOW.LIST on MAKE.GRAPH.LIST.SPEC; that is, produces a graph of this simple graph.specification as a list. Notice that selecting the title command UPDATE in this window will yield a different graph of the same structure, viz. as a graph.specification.

MAKE.GRAPH.DATUM (NODE)

Returns the DATUM associated with the graph node NODE.

MAKE.GRAPH.STATE (NODE)

Returns the STATE associated with the graph node NODE.

MAKE.GRAPH.FATHER (NODE)

Returns the graph node which is the father of the graph node NODE.

MATHFNS

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

MATHFNS contains functions from mathematics that are not already provided by Interlisp, including extra trigonometric functions, hyperbolic functions, and macros and functions for complex arithmetic. Gamma, Bessel, Error, and other special functions are planned.

NEW CONSTANTS

The following constants are defined.

E = 2.718282
EULER = .5772157
PI = 3.141593

EXTRA LOGARITHMS AND EXPONENTIATION

The following functions are made availble in addition to LOG and EXPT provided by Interlisp.

(EXP X)
: E to the power of X. The same as (ANTILOG X).

(LOG2 X)
: Same as (LOG X 2.0).

(LOG10 X)
: Same as (LOG X 10.0).

EXTRA TRIGONOMETRIC FUNCTIONS

The following functions are made available in addition to SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN, and ARCTAN2 provided by Interlisp.

(SEC X RADIANSFLG)
(CSC X RADIANSFLG)
(COT X RADIANSFLG)
(ARCSEC X RADIANSFLG)
(ARCCSC X RADIANSFLG)
(ARCCOT X RADIANSFLG)

HYPERBOLIC FUNCTIONS

The hyperbolic functions are set up similar to the trigonometric functions except that there is no RADIANSFLG. All angles are in radians.

(SINH X)
(SECH X)
(COSH X)
(CSCH X)
(TANH X)
(COTH X)
(ARCSINH X)
(ARCCOSH X)
(ARCTANH X)
(ARCCSCH X)
(ARCSECH X)
(ARCCOTH X)

COMPLEX ARITHMETIC.

Complex numbers are implemented as CONS cells. The CAR is the real part and the CDR is the imaginary part. Macros COMPLEX, REAL, IMAGINARY, and COMPLEXIFY are the best way to assemble and take apart complex numbers.

To keep things speedy, no coercion of real numbers to complex numbers or vice versa is attempted. The user must be sure to supply only complex numbers to functions which take complex args.

COMPLEX0 = (0.0 . 0.0)
Complex number 0.
COMPLEX1 = (1.0 . 0.0)
Complex number 1.
COMPLEXI = (0.0 . 1.0)
Complex number I. The square root of -1.
COMPLEXE = (2.718282 . 0.0)
COMPLEXPI = (3.141593 . 0.0)
(COMPLEX R I)
Returns complex number with real part R and imaginary part I.
(REAL Z)
Returns real part of a complex number.

(IMAGINARY Z)

Returns imaginary part of a complex number.

(COMPLEXIFY R)

Returns complex number with real part R and imaginary part 0.0. The same as (COMPLEX R 0.0).

(CONJUGATE Z)

Returns complex conjugate of Z.

(CZEROP Z)

Predicate tests if complex Z is complex 0.

(CMINUS Z)

(CDIFFERENCE Z1 Z2)

(CPLUS Z1 Z2 ... Zn)

(CTIMES Z1 Z2 ... Zn)

(CQUOTIENT Z1 Z2)

(CABS Z)

Returns real absolute value of Z.

(CARG Z)

Returns real argument of complex Z.

(CLOG Z)

Complex natural logarithm of complex Z.

(CEXP Z)

Complex E to the power of Z. Same as (CANTILOG Z).

(CSQRT Z)

Complex square root of complex Z.

COMPLEX TRIGONOMETRIC FUNCTIONS.

There is no RADIANSFLG arg to any of the complex trigonometric functions. All angles are in radians.

(CSIN Z)

(CCOS Z)

(CTAN Z)

(CCOT Z)

(CSEC Z)

(CCSC Z)

(CARCSIN Z)

(CARCCOS Z)

(CARCTAN Z)

(CARCCSC Z)
(CARCSEC Z)
(CARCCOT Z)

COMPLEX HYPERBOLIC FUNCTIONS.

There is no RADIANSFLG arg to any of the complex hyperbolic functions. All angles are in radians.

(CSINH Z)
(CSECH Z)
(CCOSH Z)
(CCSCH Z)
(CTANH Z)
(CCOTH Z)
(CARCSINH Z)
(CARCCOSH Z)
(CARCTANH Z)
(CARCCSCH Z)
(CARCSECH Z)
(CARCCOTH Z)

MM1201

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

MM1201 is a set of functions which allow you to easily control the operation of a Summagraphics MM1201 digitizing tablet connected to your D0 via the printer port. Be sure you have RS232.DCOM loaded first. To use MM1201, load <Lispusers>MM1201.DCOM and call (MM1201). A menu will appear listing the operations you can perform. You can set the digitizer's operating mode, resolution, roundoff, or scale. You can also initialize the tablet, run its self-test, and get single points. These functions should be fairly obvious if you have a copy of the Summagraphics manual.

There is also a small Draw type program included which can be invoked by choosing the "Draw" menu item. This will open a window into which you can draw with the tablet. Push the stylus down (or hit puck button 1) to draw and push the stylus barrel button (or puck button 2) to erase. Exit with ↑E. The Draw window contains a menu in the lower left corner which allows you to select any of several predefined brushes, or sweep out any area of the screen to create a new brush ("Newbrush"). You can also draw in several modes, currently freehand or straight lines (curves is not implemented). You can paint in paint or invert modes (standard bitblt meaning).

MOVE-WINDOWS

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

MOVE-WINDOWS is a tool to help you rearrange your screen quickly. Once loaded, buttoning in the background will shift you into window-moving mode. Buttoning again in the background will get you out of that mode. The cursor changes to a tile insert Graphic here when in widow-moving mode.

In window-moving mode, buttoning in a window with either the LEFT or MIDDLE button will either reshape or move the window: if you button down near a corner, the corner is moved; near a side, that side is moved; in the center, the whole window is moved. (The window is divided like a Tic-Tac-Toe board for determining these regions.) For best results, grab hold of the portion of the window you want to move (button down), drag it to the new position, and release it (button up). The cursor shifts to indicate the direction of dragging.

Buttoning in a window with the RIGHT button will call the usual window menu (DOWINDOWCOM). This is the best way to close windows, for example.

MOVE-WINDOWS requires the BACKGROUNDBUTTONEVENTFN mechanism which is new to the Harmony release.

MSPATCH

By: Mike Bird (Bird.Pasa @ Xerox)

This file can be loaded after **MASTERSCOPE** to effect the following changes:

(1) PPE checking is performed for all calls to spread functions whose argument count can be obtained. This obviates the need for many templates.

(2) The **REFERENCE** relation excludes the **SET** relation (**SMASH** and **TEST** are still included). This aids identification of "variables" which are **SET** but never **REFERENCE**'d.

MULTIMENU

By: Eric Shoen

MULTIMENU is a package which allows you to create a number of menus or windows to be associated with a main window. For example, consider an editor whose commands (menu driven) are a function of the item selected in the editor window. It is nice to have those menus which are contextually inappropriate to disappear, and to have the only appropriate menu (or menus) be highlighted in some way.

(MAKEMENUS WINDOW MENULST WIDTH)

WINDOW is the window to associate the menus with. MENULST is a list of menus of windows to associate. Each item in the list can be one of: - An instance of the record MENULSTENTRY (see below);

- An already created MENU;
- An already created WINDOW;
- A LITATOM which resolves to a MENU or WINDOW.

For each item which is a MENULSTENTRY, a menu will be created in a standard form: centered items and a 2 pixel border. If MENUSCROLLFLG is a number, and MENUITEMS is a list or a MENU, if the height of the menu is greater than MENUSCROLLFLG, the menu is placed in a scrolling window, limited to MENUSCROLLFLG pixels high.

If WIDTH is specified, all menus are created with ITEMWIDTH = WIDTH. If not specified, MAKEMENUS calls the function \FIND-WIDEST- MENU, to determine the width for created menus.

The main window gets several properties for bookkeeping; CLOSEFN and MOVEFN are set so that its subsidiary windows are also closed or moved, and RESHAPEFN becomes DON'T. The new property ITEMMENU is a list of subsidiary windows of the form (TITLE . WINDOW) where WINDOW is either a window from MENULST or the window containing a MENU from MENULST. The new property MINMENUHEIGHT keeps track of the bottom of the column of menus.

(DELETEALLMENUS MAINW)

deletes all menus associated with MAINW.

(CREATEWINDOWFROMBOX WIDTH HEIGHT TITLE)

uses GETBOXPOSITION to ghost a region WIDTH by HEIGHT, returning a window created at the final position, having title TITLE.

GRAYMENU MAINWINDOW MENUID #ITEMSTOGRAY)

grays out the menu. If #ITEMSTOGRAY is NIL, the window's BUTTONEVENTFN is stored away as another property so the menu is deactivated, and the menu is completely grayed over. If #ITEMSTOGRAY is a number, that number of menu items, counting from the bottom of the menu, going up, is grayed over. Grayed over items in such a partially grayed menus will not respond to selection.

If #ITEMSTOGRAY is 0, UNGRAYMENU is called, instead.

(UNGRAYMENU MAINWINDOW MENUID)

reactivates a (partially) grayed the menu, restoring both its appearance and original behavior.

(ADDMULTIMENU MAINW NEWMENULSTENTRY WHERE)

adds a new menu to the menucolumn. WHERE is a list of form (FIRST), (LAST), (BEFORE <MENUID>), or (AFTER <MENUID>). If NIL, WHERE defaults to (LAST).

(DELMULTIMENU MAINW MENUID)

deletes the menu with MENUID; all menus beneath the deleted one move up to fill up the blank space.

(CHANGEMULTIMENU MAINW OLDMENUID NEWMENUID)

deletes the oldmenu and replaces it with the new one. (MENUEXISTS MAINW MENUID) Returns T if there is a menu with MENUID associated with MAINW.

(MAKETTYWINDOW MAINW TTYH)

makes a TTY window associated with window MAINW. This window will be as wide as MAINW plus the WIDTH of the menucolumn associated with MAINW, and will be TTYH (default \TTYHEIGHT) high. The window is located directly below MAINW.

(WINDOWDOFUN WINDOW FN)

an NLAMBDA. Resets the TTYDISPLAYSTREAM to WINDOW (if NIL, creates one with MAKETTYWINDOW) and evaluates FN.

(\MAKEMENU WINDOW MENULSTENTRY WIDTH)

puts up the individual menus or windows.

(\FIND-WIDEST-MENU MENULST)

scans MENULST to determine the widest menu or window.

(\FIND-MENUCOLUMN-HEIGHT MENULST)

like above, only returns the height of the column of menus to be created from MENULST.

(\MENUCLOSEFN WINDOW)

is the function put on the main window's CLOSEFN property, to close the subsidiary menus/windows.

(\MENUMOVEFN WINDOW)

like above, but for the MOVEFN property. Uses RELMOVEW to retain the same relative position between the main window and subsidiary windows after the move.

The record template MENULSTENTRY is defined as (RECORD MENULSTENTRY (MENUITEMS MENUTITLE SELECTEDFN MENUID MENUSCROLLFLG)). If MENUID is NIL, it receives the value of MENUTITLE. MENUSCROLLFLG is either

NIL: Never scroll;

a NUMBER: Limit the menu to NUMBER pixels high, and make it scroll. If menu is shorter than NUMBER pixels, a standard (non-scrolling) menu is created.

MULTIW

By: Christopher Lane (Lane @ SUMEX-AIM.ARPA)

MULTIW.LSP contains the basic functions necessary for a hierarchical window environment. The functions provide both the means to link windows and the changes to the standard window functions to handle multiwindows in a reasonable way. Only the minimal support is incorporated since it is unclear how multiwindows will be used in different settings.

Windows that belong to other windows will move, close, shrink and open when their super window does. A super window is not affected by the window functions of its sub windows (but can be made to be). Sub windows can be moved, closed, shrunk, opened, and so forth as normal windows. When buried, multiwindows bury as a group, retaining their relative layering, but going below unrelated windows.

To get the full support for multiwindows, (MULTIWADVICE) must be invoked. This function advises several standard window functions to reset the window layers after they perform their operations. Multiwindows can be used without advice, but the windows will have to be explicitly reset. Currently the functions being advised are (BURYW CLEARW MOVEW OPENW SHRINKW). Where possible, functions on the windows property list were changed rather than advising the existing ones, to keep the system as compatible as possible with any environment its used in.

If SuperWindow, or SubWindowLst are NIL in following functions, they are prompted for with \MULTIWPROMPT.

MULTIW (SuperWindow SubWindowLst) [*function*]

SubWindowLst can be a list of windows or a window. MULTIW adds windows in SubWindowLst to SuperWindow's SUBWINDOWLST property, if it exits and creates it otherwise (WINDOWADDPROP). Also, SuperWindow gets added to the SUPERWINDOWLST property of each element of the SubWindowLst, providing pointers in both directions (currently, the multiwindow package does not use the backward pointers, they are provided for the user's convienence). Superwindow also gets its window function (see \MULTIWNEWFNS) properties redefined to the multiwindow versions.

WARNING

Window structures must be DAGs (directed acyclic graphs), there should be no way a window can reach itself by following out the pointers to its subwindows Recursive window structures will cause infinite looping of the window functions.

MULTIWFREE (SuperWindow SubWindowLst) [*function*]

SubWindowLst can be a list of windows, a single window, T or NIL. MULTIWFREE frees the links from SuperWindow to the windows in SubWindowLst as well as the pointers back If SubWindowLst is T, then SuperWindow is freed of all its subwindows (and they of it).

MULTIWADVICE () [*function*]

MULTIWADVICE advises functions based on the scripts in \MULTIWADVISEDFNS. The function returns a list of the advised functions which should be saved if one wishes to UNADVISE the functions (which is recommended before another call to (MULTIWADVICE)).

MULTIWGRAPH (SuperWindow DisplayWindow) [*function*]

MULTIWGRAPH graphs a window relation structure in DisplayWindow (which can be NIL). Useful in debugging relationships between windows. The GRAPHER package must be loaded to use this function.

\MULTIWGRAPH (NodeWindow) [*function*]

Function MULTIWGRAPH uses to recursively follow down the window links.

MultiWindow Window Functions

Most are functionally identical to the normal window functions except they perform the proper task on all of the subwindows and reset the window structure.

\MULTIWMOVEFN (Window NewPosition) [*function*]

\MULTIWCLOSEFN (Window) [*function*]

\MULTIWSHRINKFN (Window) [*function*]

\MULTIWEXPANDFN (Window) [*function*]

\MULTIWTOTOPFN (Window) [*function*]

\MULTIWBURYFN (Window) [*function*]

\MULTIWPROMPT (PromptString) [*function*]

Prints PromptString in PROMPTWINDOW and returns a (WHICHW (GETPOSITION))

NOTE

A demonstration of multiwindows can be found in MULTIW.DEMO. The GRAPHER package should be loaded, as well as MULTIW.LSP. Then call (DEMO MULTIWDEMO) starts the demo.

NOTEPAD

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

NOTEPAD is a package for creating NOTEPAD windows - windows in which one can do artwork at the bitmap level. The ideas in this package come pretty directly from other people's work, both inside and outside Xerox, including Markup, Draw and Smalltalk. Notepad as it stands is the product of about a man-week of work, using standard Interlisp-D as released and the EDITBITMAP package of bitmap manipulation functions. It provides an interface to some distinctly interesting functionality. Comments and suggestions are welcomed.

NOTEPAD (BITMAP COLORFLG)

Creates a NOTEPAD window. If BITMAP is NIL, then you are prompted for region for the window. Otherwise, a region is defined from the size of BITMAP, and you are prompted to move it to a desired position. If COLORFLG is non-NIL, the NOTEPAD window will be used only for control (for menu's, etc.), and all the painting will take place on the color screen.

There are two menus: one in the title of the window, and one in the window proper. Both are invoked by buttoning with either left or middle button.

Title menu

This menu gives access to commands for manipulating the window as a whole:

HELP

New.Notepad

Copy.Notepad

Save.as, bitmap.

Inspect.STYLE

Window menu

This menu has two columns of commands: those in the left column of the menu are for painting/erasing material into/from the bitmap (if this menu is invoked with the left button, painting is implied; if with the middle button, erasing); those in the right column of the menu are for changing the style in which the operations work.

Painting/Erasing

You can paint/erase using trajectories, or using (or editing) single objects. The commands in the left column of the menu are divided into two sets which reflect this division.

Trajectories

Sketch	Follows the mouse to define a trajectory of points at which to sketch.
Line	Prompts for endpoints (fix and rubberband) and uses the points on the line as a trajectory.
Circle	Prompts for center and a point on the circumference, and uses the points on the line as a trajectory.
Ellipse	Prompts for center, end of semi-major axis and end of semi-minor axis, and uses the points on the line as a trajectory.
Open curve	Prompts for first point and one or more subsequent points. You indicate that you are finished by depressing the left shift key on the last point defining the curve. A smooth curve is fitted through these points and used as a trajectory.
Closed curve	Like open curve, except that the fitted curve is closed, starting at the last point given, and proceeding to the first and then subsequent points.

Objects/editing

Text	Prompts for text, and permits positioning it with the mouse.
Area of the screen	Prompts for a (rectangular) region of the screen and places permits placing them where you want in the window.
Shade rectangle	Prompts for a region, and paints/erases it with the current shade (a shade of black does complete paint and erase).
Fill	Prompts for a region within which to fill, and a point within the area to be filled; fills the area (not necessarily a rectangle, but defined by being closed rectilinearly) with the current shade.

Edit area — Prompts for a region of the window and invokes the (Trillium) bitmap editor (standard Interlisp-D bitmap editor is the "hand-edit" choice on the submenu; others allow reflecting, rotating, shifting, inverting, putting on borders, etc.) on it. If the bitmap resulting from the edit is the same size as the original, it replaces the original region; if not, you are promtped for a place to put it.

Style

Notepad operations (see above) are carried out in a style. The style at any given moment is given by a collection of characteristics. The current style can be saved (use the command SAVE.STYLE) under a name and restored (RESTORE.STYLE). Styles may be deleted (DELETE.STYLE) from the collection of saved styles. The style collection is currently part of notepad. Consequently, moving styles between loadups is not directly supported. (It is always possible to save it on a separate file. The styles are stored as the value of NOTEPAD.STYLES. It includes bitmaps, and msut therefore be added as an UGLYVARS.)

The characteristics in the style are:

Brush — A bitmap which is either painted or erased at each point on or resulting from (see symmetry) a trajectory (see the operations paint, line, circle, ellipse). DEFINE.BRUSH prompts for a region which will then become the brush. EDIT.BRUSH permits editing of the brush bitmap (using the same editor as the operation EDIT.AREA - see above). BRUSH = COOKIE.CUT.WITH.MASK also defines a brush (see mask, below).

Use mask — An indication of whether of not to use the masking function (see mask, below) before painting/erasing. USE.MASK toggles this setting.

Mask — A bitmap which is used to clear out an area before the brush is used. The mask is erased if the brush is painting, and visa-versa. DEFINE.MASK prompts for a region which will then become the mask. EDIT.MASK permits editing of the mask bitmap (using the same editor as the operation EDIT.AREA - see above). MASK = OUTLINE.OF.BRUSH defines the mask to be the same size as the brush, with a pattern (of black) which is the filled-in outline of the brush. BRUSH = COOKIE.CUT.WITH.MASK allows you to move the mask over a section of the window and define the brush as the points so covered.

Use grid	An indication of whether or not to use the gridding function (see grid, below) while painting/erasing. USE.GRID toggles this setting.
Grid	An origin and the point (1, 1) of a grid to be used to "attract" the points used in following a trajectory. DEFINE.GRID prompts for the origin and (1, 1) point of the grid.
Use symmetry	An indication of what sort of symmetry function to use while painting/erasing. USE.SYMMETRY permits setting it as you choose. The choices are none, left/right, up/down, 4-fold (both left/right and up/down) and 8-fold (4-fold plus reflecting about the 45-degree diagonals). The brush/mask used when painting/erasing symmetrically can themselves be either identical to the brush/mask in use or symmetrically reflected. USE.SYMMETRIC.BRUSH/MASK toggles this setting.
Point of symmetry	The point with respect to which the symmetry functions are defined. POINT.OF.SYMMETRY prompts for this point.
Text font	The font in which text is printed. DEFINE.FONT permits choosing one of the fonts already loaded or OTHER (in which case you can type in the font description (family size face)).
Shade	The shade used for the rectangle and fill operations. EDIT.SHADE permits you to edit the shade (using the standard Interlisp-D shade editor).

PATCHUP

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

PATCHUP is a LISPUSERS package which facilitates multiuser interaction on a system. It is designed to allow dumping of a patch file containing changes from a number of files. One calls:

(PatchUp filesOrEditFlg patchFileName)

If filesOrEditFlg = a list of file names, then changes which have been made to items on those files will be dumped on patchFileName. Also included will be any changes that have not been associated with any file.

If filesOrEditFlg = T, then a COMS list of all changes in the system will be computed, and the user will be left in the editor editing this COMS list. Upon exiting from the editor, the revised COMS list will be used for dumping.

If patchFileName is not given, then a file name of the form

<date>-<userName>.PATCH

is used. For example,

(PatchUp LOOPSFILES)

could dump all changes made to items on LOOPSFILES to a file named

9-NOV-82-BOBROW.PATCH