# GRAPHER

Grapher contains a collection of functions and an interface for laying out, displaying, and editing graphs, that is, networks of nodes connected by links. Graphs have node labels but not link labels. Links are drawn by default as straight lines without arrowheads, but you can control the appearance of individual links. Node labels can be single lines of text, bitmaps of arbitrary size, or image objects. Facilities exist for calling functions at the nodes in a graph, and image objects containing graphs can be constructed so you can include graphs in documents and other image structures.

For instance, the Browser module uses graphs to represent function-calling structures (from MasterScope). Such a partially specified node list need have only the graph labels and the links specified. It is given to the LAYOUTGRAPH function along with some formatting information. LAYOUTGRAPH is a Grapher function which assigns a position to each node. There are formats for laying out trees, lattices, and cyclic graphs. LAYOUTGRAPH returns an instance of the GRAPH record, which is usually given to the function SHOWGRAPH. SHOWGRAPH displays a graph in a window.

## Installation

Load GRAPHER.LCOM from the library.

## User Interface

A typical way to use Grapher is to implement a function that creates a partially specified list of graph nodes representing some user data (or control) structure. Then you can use Grapher to display and manipulate or explore that structure.

Displayed graphs can be edited using the right button on the mouse. Nodes can be added, deleted, moved, enlarged, or shrunk. Links can be added or deleted.

Displayed graphs are often used as menus: selecting a node with the left or middle button can cause user-provided functions to be called on that node.

## Functions

Grapher functions perform the following tasks:

- Creating a graph
- Laying out a graph for display
- Displaying a graph
- Editing a graph
- Inserting a graph into a document

These tasks are described in the following subsections. An additional subsection describes Grapher functions that perform other tasks.

## Creating a Graph

Start by creating a list of nodes for the graph. You can create them directly (see the GRAPHNODE Record section), or you can use the NODECREATE function.

(NODECREATE *ID LABEL POSITION TONODEIDS FROMNODEIDS FONT BORDER LABELSHADE*) [Function]

This function returns a GRAPHNODE record. The arguments of this function are the same as the corresponding fields of the GRAPHNODE record, as follows:

| **Argument** | **GRAPHNODE Field** |
|---|---|
| *ID* | NODEID |
| *LABEL* | NODELABEL |
| *POSITION* | NODEPOSITION |
| *TONODEIDS* | TONODE |
| *FROMNODEIDS* | FROMNODE |
| *FONT* | NODEFONT |
| *BORDER* | NODEBORDER |
| *LABELSHADE* | NODELABELSHADE |

*ID* and *LABEL* are required; *BORDER* defaults to 0, *LABELSHADE* defaults to WHITESHADE, and *POSITION* defaults to wherever it seems convenient.

You need to specify how the nodes are connected by providing a list of *TONODEID*S and *FROMNODEID*S for each node.

## Laying Out a Graph for Display

(LAYOUTGRAPH *NODELST ROOTIDS FORMAT FONT MOTHERD PERSONALD FAMILYD*) [Function]

Lays out a partially specified graph by assigning positions to its graph nodes. It returns a GRAPH record suitable for displaying with SHOWGRAPH. An example appears after the description of the arguments.

*NODELST* Is a list of partially specified GRAPHNODEs: only their NODELABEL, NODEID, and TONODE fields need to be filled in. NODEFONT fields may also contain font specifications to be used instead of the default supplied by the *FONT* argument. These optional fields are filled in appropriately if they are NIL. All other fields are ignored and/or overwritten.

*ROOTIDS* Is a list of the node identifiers of the nodes that become the roots.

The rest of the arguments are optional and control the format of the layout.

*FORMAT* Controls the layout of the graph. It is an unordered list of atoms or lists. The following options control the structure of the graph:

- COMPACT, the default, which lays out the graph as a forest (that is, a set of disjoint trees) using the minimal amount of screen space.

- FAST, which lays out the graph as a forest, sacrificing screen space for speed.
- LATTICE, which lays out the graph as a directed acyclic graph, that is, a lattice.

In addition, the following options control the direction of the graph:

- HORIZONTAL, the default, has roots at the left and links that run left-to-right.
- VERTICAL has roots at the top and links that run top-to-bottom.

The directions can be reversed by including the atom REVERSE in *FORMAT*.

For example:

- *FORMAT*=(LATTICE HORIZONTAL REVERSE) lays out horizontal lattices that have the roots on the right, with the links running right-to-left.
- *FORMAT*=(VERTICAL REVERSE) lays out vertical trees that have the roots at the bottom, with links running bottom-to-top.
- *FORMAT*=NIL lays out horizontal trees that have the roots on the left.

LAYOUTGRAPH creates virtual graph nodes to avoid drawing a tangle of messy lines in cases where the graph is not a forest or a lattice to begin with. It modifies the nodes of NODELST, which may involve changing some of the TONODEs fields to point to new nodes. The modified NODELST is set into the GRAPHNODEs field of a newly created GRAPH record, which is returned as the value of LAYOUTGRAPH. The creation of virtual nodes depends on whether LATTICE is a member of *FORMAT*.

In a forest, nodes are laid out by traversing the forest top-down, depth-first. If a node already has been laid out, LAYOUTGRAPH creates a copy of the node (the same NODELABEL, different NODEID, and no TONODEs), lays it down, and marks both it and the original node by setting their NODEBORDER fields and NODELABELSHADE fields. This occurs instead of drawing a link that might cut across arbitrary parts of the graph. Hence, a marked node occurs at least twice in the forest.

The default for marked nodes is to leave the shade alone and set the border to 1. To alter this appearance, add the (MARK . PROPS) to the FORMAT argument. PROPS is a property list. If it is NIL, marking is suppressed altogether. If it contains BORDER or LABELSHADE properties, those values are used in the corresponding fields of marked nodes. For example, a format of (MARK BORDER 5) would cause duplicated nodes to be boxed with borders 5 points wide.

*FORMAT* adds a few enhancements to this basic marking strategy, and can include one or both of these atoms:

- COPIES/ONLY—Only the new virtual nodes are marked. The original is left unmarked.
- NOT/LEAVES—Marking is suppressed when the node has no daughters.

For example:

- *FORMAT*=(COPIES/ONLY NOT/LEAVES) marks nodes that are copies of nodes that have daughters (for example, if you see a mark, the node has daughters that are not drawn).
- *FORMAT*=(NOT/LEAVES) marks both copies and originals, but only when they have daughters.
- *FORMAT*=NIL marks originals and copies regardless of progeny.

If *FORMAT* includes LATTICE, then a node that is the daughter of more than one node is not marked. Instead, links from all its parents are drawn to it. No attempt is made to avoid drawing lines through nodes or to minimize line crossings. However, in HORIZONTAL format, nodes are positioned so that From is always left of To.

Similar conventions hold for the other formats. In VERTICAL format, for instance, the TONODEs of a node are positioned beneath it, and the FROMNODEs are positioned above it.

Cyclic graphs cannot be drawn using this convention, since a node cannot be left of itself. When LAYOUTGRAPH detects a node that points to itself, directly or indirectly, it creates a virtual node, as described above, and marks both the original and the copy. If *FORMAT* includes COPIES/ONLY, then only the newly created node is marked.

*FONT* Is a font specification for use as the default NODEFONT.

The remaining three arguments control the distances between nodes. NILs cause "pretty" defaults based on the size of *FONT*.

*PERSONALD* Is specified in points; it controls the minimum distance between any two nodes.

*MOTHERD* Is the minimum distance between a mother and her daughters.

*FAMILYD* Controls the minimum distance between nodes from different nuclear families. The closest two sister nodes can be is *PERSONALD*. The closest that two nodes that are not sisters can be is *PERSONALD+FAMILYD*.

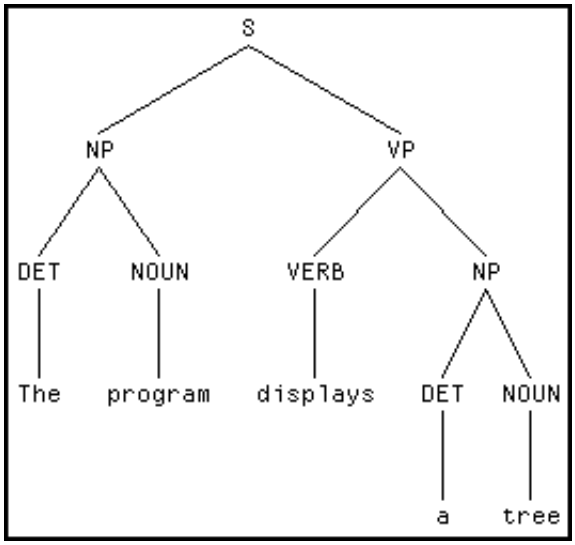
LAYOUTGRAPH reads but does not change the fields NODEBORDER and NODELABELSHADE of the nodes given it. The marked nodes are an exception. Thus, if you plan to install black borders around the nodes after the nodes have been laid out (for example, by RESET/NODE/BORDER, described in the Performing Other Tasks section), it is a good idea to give LAYOUTGRAPH nodes that have white borders. This causes the nodes to be laid

out far enough apart that when you blacken the borders later, the labels of adjacent nodes are not overwritten.

As an example, to create and display the following parse tree for the sentence "The program displays a tree.", enter the following:

```
(SETQ Snode (NODECREATE 'S 'S NIL '(NP1 VP)))
(SETQ NP1node (NODECREATE 'NP1 'NP NIL '(DET1 NOUN1) '(S) ))
(SETQ DET1node (NODECREATE 'DET1 'DET NIL '(THE) '(NP1) ))
(SETQ THEnode (NODECREATE 'THE 'The NIL NIL '(DET1) ))
(SETQ NOUN1node (NODECREATE 'NOUN1 'NOUN NIL '(PROGRAM) '(NP1) ))
(SETQ PROGRAMnode (NODECREATE 'PROGRAM 'program NIL NIL '(NOUN1) ))
(SETQ VPnode (NODECREATE 'VP 'VP NIL '(VERB NP2) '(S) ))
(SETQ VERBnode (NODECREATE 'VERB 'VERB NIL '(DISPLAYS) '(VP) ))
(SETQ DISPLAYSnode (NODECREATE 'DISPLAYS 'displays NIL NIL '(VERB) ))
(SETQ NP2node (NODECREATE 'NP2 'NP NIL '(DET2 NOUN2) '(VP) ))
(SETQ DET2node (NODECREATE 'DET2 'DET NIL '(A) '(NP2) ))
(SETQ Anode (NODECREATE 'A 'a NIL NIL '(DET2) ))
(SETQ NOUN2node (NODECREATE 'NOUN2 'NOUN NIL '(TREE) '(NP2) ))
(SETQ TREEnode (NODECREATE 'TREE 'tree NIL NIL '(NOUN2) ))

(SHOWGRAPH (LAYOUTGRAPH (LIST Snode NP1node DET1node THEnode NOUN1node
PROGRAMnode VPnode VERBnode DISPLAYSnode NP2node DET2node Anode NOUN2node
TREEnode ) '(S) '(VERTICAL)))
```



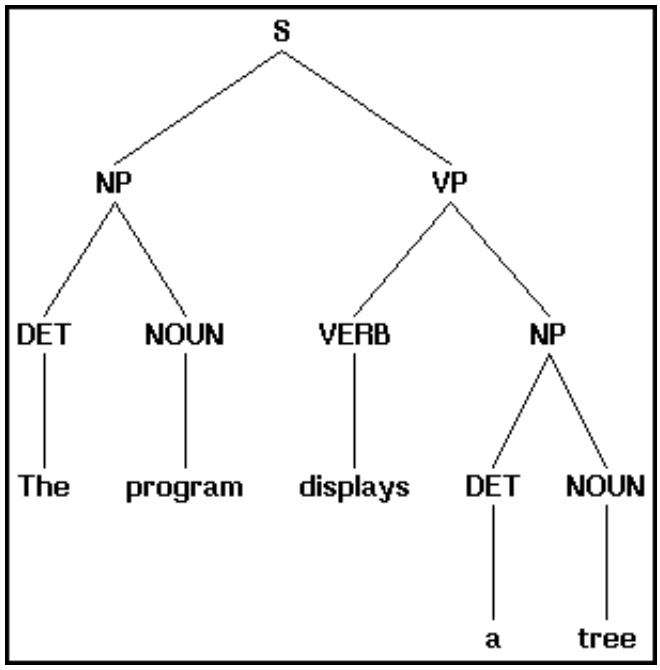

(LAYOUTSEXPR *SEXPR FORMAT BOXING FONT MOTHERD PERSONALD FAMILYD*) [Function]

Is just like LAYOUTGRAPH, except it gets its graph as an s-expression rather than a list of GRAPHNODEs. Its first argument is recursively interpreted as follows: If the s-expression is a non-list, its NODELABEL is itself and it has no TONODEs; else its CAR is taken as its NODELABEL and its CDR, which must be a list of s-expressions, is taken as its TONODEs.

Note: Circular s-expressions are allowed.

For example, to display the following parse tree for the sentence "The program displays a tree.", enter:

```
[SHOWGRAPH (LAYOUTSEXPR '(S (NP (DET The)(NOUN program)) (VP
(VERB displays) (NP (DET a)(NOUN tree)))) '(VERTICAL) NIL
'(HELVETICA 12 BRR]
```




## Displaying a Graph

(SHOWGRAPH *GRAPH W LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG ALLOWEDITFLG COPYBUTTONEVENTFN*) [Function]

Displays the nodes in *GRAPH*.

If *W* is a window, the graph is displayed in it. If the graph is larger than the window, the window is made a scrolling window. If *W* is NIL, the graph is displayed in a window large enough to hold it. If *W* is a string, the graph is displayed in a window large enough to hold it, and the window uses the string for the window title. The graph is stored on the GRAPH property of the window. SHOWGRAPH returns the window.

If either *LEFTBUTTONFN* or *MIDDLEBUTTONFN* is non-NIL, the window is given a *BUTTONEVENTFN* that, in effect, turns the graph into a menu. Whenever you press left or middle mouse button and the cursor is over a node, that node is displayed inverted, indicating that it is selected. Releasing the mouse button calls either the *LEFTBUTTONFN* or the *MIDDLEBUTTONFN* with two arguments: the selected node and the window. The node is a GRAPHNODE, or NIL if the cursor was not over a node when the button was released. The function can access the graph via the window's GRAPH property.

The graph's initial position in the window is determined by *TOPJUSTIFYFL*. If T, the graph's top edge is positioned at the top edge of the window; if NIL, the graph's bottom edge is positioned at the bottom edge of the window.

*ALLOWEDITFLG* and *COPYBOTTONFLG* are described under "Editing a Graph," below.

Note: The node labels are reprinted whenever the graph is redisplayed. If this makes scrolling of a large graph unacceptably slow, some speedup may be achieved by instructing Grapher to cache bitmaps of the labels with the nodes so they can be rapidly BITBLTed to the screen (set the variable CACHE/NODE/LABEL/BITMAPS to T). The possible gain in time,

however, may be offset by the increased storage required for the cached bitmaps.

(DISPLAYGRAPH *GRAPH STREAM CLIP/REG TRANS*) [Function]

Displays the specified graph on *STREAM*, which can be any image stream, with coordinates translated to *TRANS*. Some streams might also implement *CLIP/REG* as a clipping region. This is primarily to improve efficiency for the display.

(HARDCOPYGRAPH *GRAPH/WINDOW FILE IMAGETYPE TRANS*)
[Function]

Produces a file containing an image of *GRAPH*, that is, like SHOWGRAPH, but for files. If *GRAPH/WINDOW* is a window, HARDCOPYGRAPH operates on its GRAPH window property. The *FILE* and *IMAGETYPE* argument are given to OPENIMAGESTREAM to obtain a stream on which the graph is displayed. *TRANS* is the position in screen points (that is, it is scaled by the image stream's DSPSCALE) of the lower-left corner of the graph relative to the lower-left corner of the piece of paper.

GRAPH/HARDCOPY/FORMAT [Variable]

Is used to control the format of the graph when printing to paper. It is a property list that contains the following properties.

- MODE

  Determines the orientation of the hardcopy of the graph. The value can be LANDSCAPE, or PORTRAIT (the default). If LANDSCAPE, the graph is shown with the longer paper edge as the major axis. If PORTRAIT, graph is shown with the shorter paper edge as the major axis. If you use the window menu command to hardcopy, the graph is shown in PORTRAIT mode.

- PAGENUMBERS

  Determines whether to print the page number. The value can be T or NIL. If T, GRAPHER prints the page number in X-Y format on the upper right corner of each page. If NIL, no page number is printed.

- TRANS

  Determines where to position the graph on paper. The value can be NIL or a position. If NIL, each graph is positioned at the center of the paper. If a position, GRAPHER determines the location in screen points of the lower left corner of the graph relative to the lower left corner of the paper.

The initial value of GRAPH/HARDCOPY/FORMAT is set to
(MODE PORTRAIT PAGENUMBERS T TRANS NIL)

DEFAULT.GRAPH.WINDOWSIZE [Variable]

Contains a list of two numbers in screen points. The first number indicates the window width. The second number indicates the window height. This variable is used to control the maximum size of a graph window when it first gets displayed.

## Editing a Graph

If *ALLOWEDITFLG* in the SHOWGRAPH function is non-NIL, you can position the cursor over a node and use the right mouse button to edit the graph. (The normal window commands can be accessed by right-clicking (pressing the right mouse button) in the border or title regions.) Holding down the CONTROL key and simultaneously pressing the right mouse button allows you to position nodes by tracking the cursor. Pressing the right mouse button without the control key pops up the following menu of edit operations.


The edit operations allow moving, adding, and deleting of nodes and links.

- Adding a node prompts for a NODELABEL creates a new node with that label, adds it to the graph, and allows you to position it.
- Deleting a node removes it (using DREMOVE) from the graph after deleting all of the links to and from it.
- Selecting Directed or Sides allows you to control how links are drawn between nodes. See the section "Graph Record" for more information.
- Selecting Border allows you to invert the border around a node's label.
- Selecting Shades allows you to invert a node's label.

When you select the STOP menu command, the graph window is closed.

*COPYBUTTONEVENTFN* is a function to be run when you copy-select from the Grapher window. If this is not specified, the default simply COPYINSERTs a Grapher image object.

Certain fields of the GRAPH record contain functions that are called from the graph editor menu to perform actions on an element in the displayed graph. They allow the graph to serve as a simple edit interface to the structure being graphed.

The following fields of a graph contain editing functions and the arguments that are passed to those functions when they are called. In all cases, *GRAPH* is the graph being displayed, and *WINDOW* is the window in which it is displayed.

Even if you do not specify any of the following fields (except GRAPH.ADDNODEFN), the system provides a set of functions which allows you to edit the graph. Any functions you specify are executed *after* the default function is executed. For example, if you supply the ADDLINKFN and then add a link using the edit menu, your function is called after the link is added. Exception: for ADDNODEFN, the function is called and must return a node to be added to the graph; this function is executed *instead* of the default function.

GRAPH.MOVENODEFN *NODE NEWPOS GRAPH WINDOW OLDPOS* [Record field]

Contains a function that is called with the arguments shown after you have stopped moving a node interactively; that is, it is not called as the node is being moved. *NEWPOS* is the new position of the node, *OLDPOS* its original position. The difference between them can be used, for example, to move other related nodes by the same distance.

GRAPH.ADDNODEFN *GRAPH WINDOW* [Record field]

Is called when you select ADD A NODE. Returns a node, or NIL if no new node is to be added. A node-moving operation is called on the new node after it is created to determine its position.

GRAPH.DELETENODEFN *NODE GRAPH WINDOW* [Record field]

Is called when a node is deleted. Before this function is called, all of the links to or from the node are deleted.

GRAPH.ADDLINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is added.

GRAPH.DELETELINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is deleted, which can be either directly or from deleting a node.

GRAPH.FONTCHANGEFN *HOW NODE GRAPH WINDOW* [Record field]

Is called for side effect only when you ask for the label on a node to be made larger or smaller. *HOW* is either LARGER or SMALLER.

GRAPH.CHANGELABELFN *GRAPH NODE* [Record field]

Is called for side effect only when you ask to change the label of a node, for example, using EDITGRAPH.

GRAPH.INVERTBORDERFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the border of a node, for example, using EDITGRAPH.

GRAPH.INVERTLABELFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the label of a node, for example, using EDITGRAPH.

## Editing Menu

The editing menu is controlled by the following two variables:

EDITGRAPHMENU [Variable]

Contains the editing menu, if it exists, or NIL. If you press the right button and it is NIL, a fresh menu is created from EDITGRAPHMENUCOMMANDS.

EDITGRAPHMENUCOMMANDS [Variable]

A list of menu items used to create EDITGRAPHMENU. The contents of EDITGRAPHMENUCOMMANDS must be a list that can be used as the ITEMS field of a MENU; see MENU in the *Interlisp-D Reference Manual* for details.

## Inserting a Graph into a TEdit Document

A graph data structure can be encapsulated in a Grapher image object so that it can be inserted in a TEdit document or other image structure. Grapher image objects are constructed by the following function.

(GRAPHEROBJ *GRAPH HALIGN VALIGN*) [Function]

Returns a Grapher-type image object that displays *GRAPH*.

*HALIGN* and *VALIGN* specify how the graph is to be aligned with respect to the reference point in its host, for example, a TEdit file or image object window. They can be numbers between zero and one, specifying as a proportion of the width/height of the graph the point in the graph that overlays the reference point; zero means that the graph sits completely above and to the left of the reference point, and one means it sits completely below and to the right.

They can also be pairs of the form (*NODESPEC POS*), where

*NODESPEC* specifies a node that the graph is to be aligned by, and *POS* specifies where in the node the alignment point is. The *NODESPEC* can be either a NODEID or one of the atoms *TOP*, *BOTTOM*, *LEFT*, or *RIGHT*, indicating the topmost, bottommost, etc., node of the graph.

*POS* can be a number specifying proportional distances from the lower-left corner of the node, or the atom BASELINE, indicating the character baseline (for *VALIGN*, or simply zero for *HALIGN*).

For example, to align a linguistic tree so that the baseline of the root node is at the reference point, *VALIGN* is (*TOP* BASELINE).

The *BUTTONEVENTINFN* of the image object pops up a single-item menu, which, if selected, causes the graph editor to be run.

## Performing Other Tasks

Grapher functions also allow you to return the smallest region containing all nodes, invert a node region, reset fields in a node, print a graph to a stream, read a graph from a stream, and edit a graph.

(GRAPHREGION *GRAPH*) [Function]

Returns the smallest region containing all of the nodes in GRAPH.

(FLIPNODE *NODE DS*) [Function]

Inverts a region in the stream DS that is one pixel bigger all around than NODE's region. This makes it possible to see black borders after the node has been flipped.

(RESET/NODE/BORDER *NODE BORDER STREAM GRAPH*) [Function]
and
(RESET/NODE/LABELSHADE *NODE SHADE STREAM*) [Function]

Reset the appropriate fields in the node. If *STREAM* is a display stream or a window, the old node is erased and the new node is displayed. Changing the border may change the size of the node, in which case the lines to and from the node are redrawn. The entire graph must be available to RESET/NODE/BORDER for this purpose, either supplied as the *GRAPH* argument or obtained from the GRAPH property of *STREAM,* if it is a window. Both functions take the atom INVERT as a special value for *BORDER* and *SHADE*. They read the node's current border or shade, calculate what is needed to invert it, and do so.

(DUMPGRAPH *GRAPH STREAM*) [Function]

Prints *GRAPH* out on *STREAM* in a special, relatively compact encoding that can be interpreted by the function READGRAPH, below. Graphs cannot be saved on files simply by ordinary print functions such as PRIN2. This is because the Grapher functions use FASSOC (that is, EQ, not EQUAL) to fetch a graph node given its ID, so reading it back in gives the right result only if the IDs are atomic. HPRINT resolves this problem, but it tends to dump too much information: it dumps a complete description of the node font, for example, including the character bit maps. DUMPGRAPH and READGRAPH are used in the implementation of Grapher image objects.

(READGRAPH *STREAM*) [Function]

Reads information from *STREAM* starting at the current file pointer and returns a graph structure equivalent to the one that was given to DUMPGRAPH.

(EDITGRAPH *GRAPH WINDOW*) [Function]

Enables editing *GRAPH* in *WINDOW*. If *GRAPH* is NIL, an empty graph is created for editing. If *WINDOW* is NIL, a window of appropriate size is created.

(GRAPHERPROP *GRAPH PROP NEWVALUE*) [Function]

Accesses GRAPH.PROPS field of *GRAPH* record. The function returns the previous value of *GRAPH*'s *PROP* aspect. If *NEWVALUE* is given, it is stored as the new *PROP* aspect.

## Grapher Record Structure

Grapher has GRAPH records which represent graphs. Within these records are GRAPHNODE records which are lists of graph records.

### GRAPH Record

A graph is represented by a GRAPH record, which has the following fields:

```
GRAPHNODE
DIRECTEDFLG
SIDESFLG
GRAPH.MOVENODEFN
GRAPH.ADDNODEFN
GRAPH.DELETENODEFN
GRAPH.ADDLINKFN
GRAPH.DELETELINKFN
```

```
GRAPH.CHANGELABELFN
GRAPH.INVERTBORDERFN
GRAPH.INVERTLABELFN
GRAPH.FONTCHANGEFN
GRAPH.PROPS
```

GRAPHNODE is a list of graph nodes, and is described below.

DIRECTEDFLG and SIDESFLG are flags that control how links are drawn between the nodes. If DIRECTEDFLG is NIL, Grapher draws each link in such a way that it does not cross the node labels of the nodes it runs between. Often, this leaves some ambiguity, which is settled by SIDESFLG. If SIDESFLG is NIL, Grapher prefers to draw links that go between the top and bottom edges of nodes. If SIDESFLG is non-NIL, Grapher prefers to draw links between the sides of the nodes.

If DIRECTEDFLG is non-NIL, the edges are fixed, for example, always to the left edge of the To node. This can cause links to cross the labels of the nodes they run between. In this case, if SIDESFLG is NIL, the From end of the link is attached to the bottom edge of the From node; the To end of the link is attached to the top edge of the To node. If DIRECTEDFLG is non-NIL and SIDESFLG is non-NIL, the From end of the link is attached to the right edge of the From node; the To end of the link is attached to the left edge of the To node.

GRAPH.PROPS is a list in property-list format, and is accessed by the function GRAPHERPROP.

The remaining fields give you hooks into the graph editor, and are described in the section "Editing a Graph".

## GRAPHNODE Record

The GRAPHNODE record has the following fields of interest:

NODELABEL — Is what gets displayed as the node. If this is a bit map, BITBLT is used; if it is an image object, its *IMAGEBOXFN* and *DISPLAYFN* are used. Anything else is printed with PRIN3; see the *Interlisp-D Reference Manual*. Image objects can be used to give a node a larger-than-normal margin around its text label.

NODEID — Is a unique identifier. NODEIDs are used in the link fields instead of pointers to the nodes themselves, so that circular Lisp structures can be avoided. NODEIDs are often used as pointers to the structure represented by the graph.

TONODE — Is a list of NODEIDs. A link runs from the currently selected node to each node in TONODEs. Entries in this field can be used to specify properties of the lines drawn between nodes.

If an item in the TONODEs of the current node N1 is not a NODEID but rather a list of the form:

*(*LINK% PARAMETERS *TONODEID . PARAMLIST)*

then *PARAMLIST* is interpreted as a property list specifying properties of the link drawn from N1 to *TONODEID*.

Properties of *PARAMLIST* currently noticed are LINEWIDTH, DASHING, COLOR, and DRAWLINKFN. The first three are passed directly to DRAWLINE.

For example, if the TONODEs for A is:

```
((LINK% PARAMETERS B LINEWIDTH 4 DASHING (3 3))

(LINK% PARAMETERS C DASHING (5 1) COLOR 12))
```

then two dashed lines emanate from A, with the one to B having width 4 and dashing (3 3), and the one to C having the default width 1, dashing (5 1), and color (if implemented) 12.

If the property DRAWLINKFN is on the list, then its value must be a function to be called instead of DRAWLINE. It is passed all the arguments of DRAWLINE plus the *PARAMLIST* as a last argument.

For convenience, the variable LINKPARAMS is set to the constant value LINK% PARAMETERS. When DISPLAYGRAPH scales the graph to the units of a particular output stream, the properties whose names are found on SPECVAR *SCALABLELINKPARAMETERS* are also scaled.

FROMNODE — Is a list of NODEIDs. A link runs to the currently selected node from each node in FROMNODEs.

NODEPOSITION — Is the location of the center of the node (a POSITION).

NODEFONT — Specifies the font in which this node's label is displayed. It can be any font specification acceptable to FONTCREATE, including a FONTDESCRIPTOR. NODEFONT is changed by the graph edit operations Larger and Smaller. When this happens, the font family may be changed as well as the size. Default is the value of DEFAULT.GRAPH.NODEFONT (initially NIL, which specifies the system DEFAULTFONT).

NODEBORDER — Specifies the shade and width of the border around a node via the following values:

- NIL,0 — No border; equals border of width zero
- T — Black border, one pixel wide
- 1,2,3... — Black border of the given width
- -1,-2... — White border of the given width
- (W S) — Where W is an integer and S is a texture or a shade; yields a border W pixels wide filled with the given shade S; see the *Interlisp-D Referene Manual*.

Default is the value of DEFAULT.GRAPH.NODEBORDER (initially NIL).

NODELABELSHADE — Contains the background shade of the node. If this field is non-NIL, then when a node is displayed, the label area for the node is first painted as specified by this field, then the

label is printed in INVERT mode. This does not apply to labels that are bit maps or image objects. The legal values for the field are: NIL (same as WHITESHADE), T (same as BLACKSHADE), a texture, or a bitmap. Default is the value of DEFAULT.GRAPH.NODELABELSHADE, which is initially NIL.

NODEWIDTH and NODEHEIGHT — Are initially set by Grapher to be the width and height of the node's NODELABEL.

## Limitations

Grapher does not work well with packages. Because the node labels are printed with PRIN3, Grapher does not visually distinguish nodes whose labels are symbols in different packages; you do not see that fact displayed in the graph.