

## Grapher documentation

Richard R. Burton and Kurt VanLehn  
Last revised: December 27, 1983  
Copyright (c) 1982, 1983 Xerox Corporation

<LISP>LIBRARY>GRAPHER contains a collection of functions and an interface for laying out, displaying, and editing graphs (i.e., networks of nodes and links). Graphs have node labels but not link labels. Links are drawn as straight lines, without arrowheads. Node labels can be a single line of text or a bitmap of arbitrary size. There are facilities for having actions triggered by selection of nodes in graphs.

A typical way to use the grapher package is to implement a function that creates a partially specified graph that represents some user data (or control) structure. For instance, the browser package (<LISPUSERS>BROWSER) uses graphs to represent function calling structures (from Masterscope). Such a partially specified graph need only have the graph labels and the links specified. It is given to LAYOUTGRAPH, along with some formatting information. LAYOUTGRAPH assigns a position to each of the nodes. There are formats for laying out trees, lattices and cyclic graphs. LAYOUTGRAPH returns a GRAPH record, which is usually given to SHOWGRAPH. SHOWGRAPH displays a graph in a window. 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. Displayed graphs can be edited using the right button. Nodes can be added, deleted, moved, enlarged or shrunk. Links can be added or deleted.

This document will first describe the data structures, the GRAPH and GRAPHNODE records, then the layout functions, the display functions, and the editing functions.

### The Graph

A graph is represented by a GRAPH record, which has the following fields: GRAPHNODES, DIRECTEDFLG, SIDESFLG, GRAPH.MOVENODEFN, GRAPH.ADDNODEFN, GRAPH.DELETENODEFN, GRAPH.ADDLINKFN, GRAPH.DELETELINKFN and GRAPH.FONTCHANGEFN.

GRAPHNODES is a list of graph nodes, which are described below.

DIRECTEDFLG and SIDESFLG are flags which control how links are drawn between the nodes. If DIRECTEDFLG is NIL, the Grapher will draw 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, the Grapher prefers to draw links that go between the top and bottom edges of nodes. If SIDESFLG is non-NIL, it prefers to draw links between the sides of the nodes.

If DIRECTEDFLG is non-NIL, the edges are fixed (e.g., 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.

The remaining fields give the user hooks into the graph editor, which is described below.

### The Graph nodes

The GRAPHNODE record has the following fields of interest to the user:

NODELABEL - the thing that gets printed (using PRIN3) as the node.  
If this is a bitmap, bitblt is used instead of PRIN3. This allows icons of

arbitrary sizes to be used as graph labels.

**NODEID** - a unique identifier. **NODEID**'s are used in the link fields instead of using pointers to the nodes themselves so that circular Lisp structures can be avoided. **NODEID**'s are often used as pointers to the structure represented by the graph.

**TONODES** - a list of **NODEIDS**: a link runs from this node to each node in **TONODES**.

**FROMNODES** - a list of **NODEIDS**: a link runs to this node from each node in **FROMNODES**.

**NODEPOSITION** - the location of the center of the node (a **POSITION**).

**NODEFONT** - Specifies the font in which this node's label will be displayed. It can be any font specification acceptable to **FONTCREATE**, including a **FONTDESCRIPTOR**. **NODEFONT** is changed by graph edit operations **LARGER** and **SMALLER**. When this happens, the font family may be changed as well as the size.

**BOXNODEFLG** - If this field is non-NIL, the node label will have a box drawn around it.

**NODEWIDTH** and **NODEHEIGHT** - Normally, these are set by grapher to be the width and height of the node's **NODELABEL**. However, if the user provides fixnums for these fields, their values will be used instead. This feature can be used to give a node a larger-than-normal "margin" around its label.

Graphnodes can be created by the create operator from the record package or with the following function:

```
NODECREATE(ID LABEL POSITION TONODEIDS FROMNODEIDS FONT BOXED?)
```

This function returns a **GRAPHNODE** with **NODELABEL** = **ID**, **NODELABEL** = **LABEL**, **NODEPOSITION** = **POSITION**, **NODEFONT** = **FONT** and **BOXNODEFLG** = **BOXED?**.

A word on saving graphs on files: The grapher functions use **FASSOC** (i.e., **EQ** not **EQUAL**) to fetch a graphnode given its id. Hence, simply dumping a graph onto a file and reading it back in will only work correctly if the id's are atomic. Before dumping a graph, it is wise to set to **NIL** the fields that are normally calculated and filled in by the grapher functions. In particular, unless special provisions are made, the **NODELABEL** and the **NODEFONT** fields will read back in as litatoms with names like **{BITMAP}#1,23456** or **{FONTDESCRIPTOR}#7,2345** instead of real bitmaps and font descriptors.

## Laying out a Graph

```
LAYOUTGRAPH(NODELST ROOTIDS FORMAT FONT MOTHERD PERSONALD FAMILYD)
```

**LAYOUTGRAPH** "lays out" a partially specified graph by assigning positions to its graphnodes. It returns a **GRAPH** record suitable for displaying with **SHOWGRAPH**. The input **NODELST** is a list of **GRAPHNODES** that partially specifies a graph: only their **NODEID**, **NODELABEL**, and **TONODES** fields need to be filled in. Optional fields are: **NODEFONT**, **NODEWIDTH** and **NODEHEIGHT**. These optional fields will be filled in appropriately if they are **NIL**. All other fields will be ignored and/or overwritten. **ROOTIDS** is a list of the node identifiers of the nodes that will become the roots. The other arguments are optional and control the format of the layout.

FORMAT controls the geometry of the layout. It is an unordered list of atoms. There are three basic formats:

COMPACT [the default] - The graph is layed out as a forest (a set of trees) in such a way that the minimal amount of screen space is used.

FAST - The graph is layed out as a forest, sacrificing screen space for speed.

LATTICE - The graph is layed out as an acyclic directed graph (a lattice).

These basic formats come in the obvious two flavors:

HORIZONTAL [the default] - roots at left, links run left-to-right.

VERTICAL - roots at the top, links run top-to-bottom.

The directions can be reversed by including the atom REVERSE in FORMAT. Thus, 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 or not LATTICE is a member of FORMAT. The forest (i.e., non-LATTICE) case will be described first.

Nodes are layed out by traversing the forest top-down, depth-first. Whenever an already layed out node is found, instead of drawing a link to it that might cut across arbitrary parts of the graph, LAYOUTGRAPH creates a copy of the node (same NODELABEL, different NODEID, no TONODES), lays it down and "draws" boxes around both it and the original node (by setting their BOXNODEFLG fields). Hence, a box around a node means that it occurs at least twice in the forest. FORMAT adds few twists to this basic strategy of using boxes. It can include one or both of these atoms:

COPIES/ONLY - only the new "virtual" nodes are boxed.  
The original is left unboxed.

NOT/LEAVES - Boxing is suppressed when the node has no daughters.

For examples, FORMAT = (COPIES/ONLY NOT/LEAVES) will box nodes that are copies of nodes that have daughters (i.e., if you see a box, the nodes has daughters that aren't drawn). FORMAT = (NOT/LEAVES) will box both copies and originals, but only when they have no daughters. FORMAT = NIL boxes originals and copies regardless of progeny.

If FORMAT includes LATTICE, then a node that is the daughter of more than one node is not boxed. 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 can not be drawn using this convention, of course (how can a node 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 boxes both the original and the copy. If FORMAT includes COPIES/ONLY, then only the newly created node is boxed.

This ends the discussion of the `FORMAT` argument to `LAYOUTGRAPH`. The other arguments are not so complicated. `FONT` is a font specification for use as the default `NODEFONT`; if `NIL`, the font is taken from the `DEFAULTFONT` of the current `FONTPROFILE`.

The remaining three arguments control the distances between nodes. `NILs` cause "pretty" defaults based on the size of `FONT` to be used. `PERSONALD` 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`.

## Displaying and Editing a Graph

```
SHOWGRAPH(GRAPH W LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG ALLOWEDITFLG)
```

`SHOWGRAPH` displays the nodes in the `GRAPH`. If `W` is a window, the graph will be displayed in it. If the graph is larger than the window, the window is made a scrolling window. If `W` is `NIL`, the graph will be displayed in a window large enough to hold it. If `W` is a string, the graph will be displayed in a window large enough to hold it and the window will use the string for the window title. `SHOWGRAPH` caches some information in the `graphnode` fields in order to make scrolling faster. The graph is stored on the `GRAPH` property of the window. `SHOWGRAPH` returns the window.

If either `LEFTBUTTONFN` or `MIDDLEBUTTONFN` are non-`NIL`, the window is given a `BUTTONEVENTFN` that, in effect, turns the graph into a menu. Whenever the left or middle button is depressed and the cursor is over a node, that node will be displayed inverted, indicating that it is selected. Letting up on the button will call either the `LEFTBUTTONFN` or `MIDDLEBUTTONFN` with two arguments: the selected `GRAPHNODE` (this will be `NIL` if the cursor was not over a node when the button was released) and the window (from the window, the functions can access the graph via `WINDOWPROP`).

The graph's initial position is determined by `TOPJUSTIFYFLG`. 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.

If `ALLOWEDITFLG` is non-`NIL`, the right button can be used to edit the graph. (The normal window commands are available by right buttoning in the border or title regions.) Right buttoning while a shift key is down allows positioning of nodes by tracking the cursor. Right buttoning with the shift key up brings up a 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 the user to position it. Deleting a node removes it (using `DREMOVE`) from `GRAPH` after deleting all of the links to and from it. When the `STOP` menu command is selected, the graph window is closed.

Certain fields of the `GRAPH` record are functions that get called by the graph editor whenever an action is performed on an element in the displayed graph. They allow the graph to serve as a simple edit interface to the structure being graphed. Below are the fields of `GRAPH` and the arguments that the corresponding function will be called. In all cases `GRAPH` is the graph being displayed and `WINDOW` is the window in which it is displayed.

`GRAPH.MOVENODEFN(NODE NEWPOS GRAPH WINDOW)` called after the user has stopped moving a node interactively (i.e. is not called as the node is being moved.)

`GRAPH.ADDNODEFN (GRAPH WINDOW)` called when the user

selects "Add a node" and should return a NODE or NIL if no new node is to be added. Node moving operation will be called on the new node after it is create to determine its position.

GRAPH.DELETENODEFN (NODE GRAPH WINDOW) 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) called when a link is added.

GRAPH.DELETELINKFN (FROM TO GRAPH WINDOW) called when a link is deleted which can be either directly or from deleting a node.

GRAPH.FONTCHANGEFN (HOW NODE GRAPH WINDOW) called when the user asked for the label on a node to be made larger or smaller. HOW is one of LARGER or SMALLER.

## Miscellaneous functions

GRAPHREGION(GRAPH)

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

LAYOUTSEXPR(SEXPR FORMAT FONT MOTHERD PERSONALD FAMILYD)

Just like LAYOUTGRAPH except it gets its graph as a s-expression rather than a list of GRAPHNODEs. A formal recursive definition of its first argument is: If the s-expr is an atom, its NODELABEL will be itself and it will have no TONODEs; else it is a list and its car will be taken as its NODELABEL and its cdr, which must be a list of s-exprs, will be taken as its TONODEs. Note that circular s-expressions are allowed. For example, the following displays a parse tree for the sentence "The program displays a tree."

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

## Recent Changes

\* GRAPHER: Extensively revised; new function HARDCOPYGRAPH; node formatting extended

GRAPHER has been extensively revised, so that it uses much less memory space per node. Whereas the old Grapher created a bitmap per node, the new one doesn't. The price is that scrolling may take a little longer. To REDISPLAYW a very large graph takes twice as long as it used to (if you don't like this, set CACHE/NODE/LABEL/BITMAPS/FLG to T). Also, the GRAPHRECORD was changed to use half as many cons cells. This version will not run in Carol or older <lispcore> systems if the user depends on nodefonts being defaulted to the DEFAULTFONT font class.

(HARDCOPYGRAPH GRAPH/WINDOW FILE IMAGETYPE TRANS)

Produces a file from a formatted graph (e.g., like SHOWGRAPH, only for files). If GRAPH/WINDOW is a window, HARDCOPYGRAPH will operate on the GRAPH property of the window. If the device field of the file name is LPT, the file will automatically get sent to the appropriate printer. IMAGETYPE is either PRESS or INTERPRESS, and defaults to INTERPRESS. TRANS is a position in screen points of the lower left corner of the graph from the lower left corner of the piece of paper.

**(DISPLAYGRAPH GRAPH STREAM CLIP/REG TRANS)**

Put 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 an efficiency hack for the display.

GRAPHER now allows nodes to be "boxed" with borders of arbitrary shades and widths. Borders work for regular labels and bitmap labels, but not for imageobject labels. The old graphnode field BOXNODEFLG has been renamed NODEBORDER. It takes the following values:

NIL no border, as before  
 T black border, 1 pixel wide, as before  
 0 no border  
 1,2,3... black border of the given width  
 -1,-2... white border of the given width  
 (w s) where w is a fixp and s is a texture or a shade; yields a border w wide filled with the given shade s.

A new graphnode field, NODELABELSHADE, contains the background shade of the node. This allows GRAPHER to remember when a node is inverted. When a node is displayed, the label area for the node is first painted as specified by NODELABELSHADE, then the label is printed in INVERT mode. This does not apply to labels that are bitmaps or image objects. The legal values for the field are: NIL (same as WHITESHADE), T (same as BLACKSHADE), a texture, or a bitmap.

**(RESET/NODE/BORDER <node> <border> <stream> <graph>)****(RESET/NODE/LABELSHADE <node> <shade> <stream>)**

These functions reset the appropriate fields in the node. If <stream> is a displaystream or a window, the old node will be erased and the new node will be displayed. Changing the border may change the size of the node, in which case the lines to and from the node will be 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 would be needed to invert it, and do so.

LAYOUTGRAPH previously used a 1-pixel black box to mark certain nodes in order to indicate where it had snapped links. That is still the default action. However, the appearance of marked nodes can be controlled by adding (MARK ....) to the FORMAT argument of LAYOUTGRAPH. The tail of (MARK ....) is a property list. If the property list is NIL, marking is suppressed altogether. If a BORDER property is specified, the value will be used as the NODEBORDER of marked nodes. If a LABELSHADE property is specified, its value will be used on the marked nodes. Of course, you can specify both a BORDER and NODELABEL property.

LAYOUTGRAPH will read, but not change, the fields NODEBORDER and NODELABELSHADE of the nodes given it (except for the marked nodes, of course). Thus, if one is planning on installing black borders around the nodes after the nodes have been layed out, its a good idea to give LAYOUTGRAPH nodes that have white borders. This will cause the nodes to be layed out far enough apart that when you blacken the borders later, the labels of adjacent nodes will not be overwritten.

When a graphnode is created by the record package, the default values are now taken from the value of the following variables: DEFAULT.GRAPH.NODEBORDER, DEFAULT.GRAPH.NODELABELSHADE, and DEFAULT.GRAPH.NODEFONT. GRAPHER initializes these to NIL. To get the benefits of this new feature, the user will have to recompile functions that create graphnodes

FLIPNODE now inverts a region that is 1 pixel bigger all around than the node's region. This makes it possible to see black borders after the node has been flipped.

LAYOUTGRAPH takes a new format token. Adding REVERSE/DAUGHTERS to the list of format items will reflect horizontal graphs vertically, and vertical graphs horizontally.