

# **Lapidary Reference Manual**

**Brad T. Vander Zanden  
David Bolt**

September 1993

## **Abstract**

This document describes the features and operations provided by Lapidary, a graphical interface builder that allows a user to pictorially specify all graphical aspects of an application and interactively create much of the behavior. Lapidary allows a user to draw most of opal's objects, combine them into aggregadgets, align them using iconic constraint menus or custom constraints, and create behaviors by entering appropriate parameters in dialog boxes representing each of Garnet's interactors, or by demonstrating the appropriate behavior for feedback objects.

Copyright © 1993 - Carnegie Mellon University

This research was sponsored by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597. Additional support for Garnet has been provided by NEC, Apple, Adobe, and General Electric.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

# 1. Getting Started

To load Lapidary, type `(load garnet-lapidary-loader)` after Garnet has been loaded, or type `(defvar load-lapidary-p t)` before Garnet is loaded, and Garnet will automatically load Lapidary when the Garnet loader file is invoked. To start Lapidary, type `(lapidary:do-go)`. This will cause Lapidary to come up in its initial state with the following windows:

- editor-menu: This menu contains a set of functions that deal with aggregadgets, constraints, saving and restoring objects, deleting objects, and setting properties of objects.
- shapes menu: This menu allows the designer to create opal graphical objects and windows.
- box-constraint menu: This menu allows the designer to attach constraints to an object that control its left, top, width, and height.
- drawing window: This window allows the designer to create new objects or load objects from existing files.

## 1.1. Object Creation

Lapidary allows new objects to be created from scratch, loaded from pre-defined gadgets files, or created directly in Garnet and then linked to a Lapidary window. The shapes menu displays the primitive graphical objects that can be created in Lapidary.



**Figure 1-1:** Shapes menu

The first six geometric shapes can be created by selecting the appropriate menu-item and sweeping out the item in a drawing window with the right mouse button down. Feedback corresponding to the selected shape will be shown as the object is swept out. Properties such as line-style, filling-style, and draw-function can be set from the corresponding property menus (see section 2.3).

To create a single line of text, select text and then click where you want the text to start. A cursor will

appear and one line of text can be entered from the keyboard. For more than one line of text use multi-text. Single-line text can be terminated with either a mouse click or by hitting RETURN but, multi-line text can only be terminated by a mouse click.

To create a window, select the window menu-item and Lapidary will create a new window. Since, new windows initially have the same size and location as the draw window, they must be moved in order to expose the original draw window.

Bitmaps can be loaded by selecting the bitmap menu-item. Lapidary brings up a dialog box that allows the user to enter the name of an image file and the window that the bitmap should be placed in. The window name is obtained from the title border that surrounds a window or the name that appears in the icon for the window.

To create a horizontal or vertical list, first select a prototype object. Then select horizontal or vertical list and sweep out the list. A property sheet will appear that can be used to set parameters that control the list's appearance. A description of the parameters can be found in the chapter on aggregadgets and aggreglists.

## 1.2. Selecting Objects

Lapidary permits two types of selections: primary selections and secondary selections. Primary selections are denoted by black grow boxes that sprout around the perimeter of an object; secondary selections are denoted by white grow boxes. Most operations do not distinguish between these two types of selections and will operate in the same way on both types of selections. However, two operations, attaching a constraint to an object and defining parameters for an object, do make this distinction.

Lapidary provides the usual range of selection operations including selecting an object, deselecting an object, and adding new selections. In addition, Lapidary allows the user to select covered objects by pointing at an already selected object and requesting that the object directly covered by the selected object be selected. Finally Lapidary allows the user to select an object's aggregadget (i.e., its parent in the aggregadget hierarchy) by pointing at a selected object and selecting it again (using the selection button, not the add to selection button). Lapidary will cycle around to a leaf if the selected object is at the top-level of the aggregadget hierarchy.

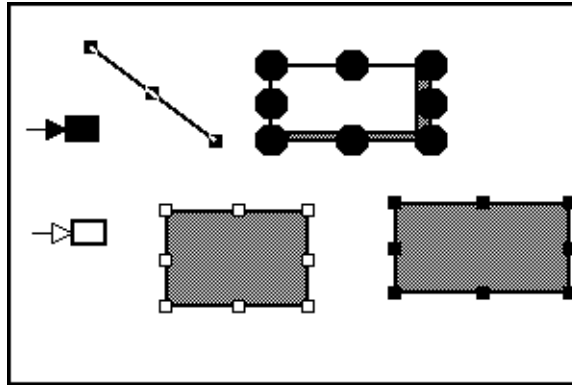
The type of grow boxes that appear around the perimeter of an object depend on the type of object selected. Primitive opal objects sprout rectangular grow boxes around their perimeters whereas aggregadgets sprout circular grow boxes (Figure 1-2). If an object is too small to accommodate all eight grow boxes (whether it is a primitive object or aggregadget object), then an arrow appears that points at the object.

Section 1.3 provides specific details on each of the selection operations.

## 1.3. Mouse-Based Commands

Lapidary is primarily a mouse-based system so it is important to know which mouse buttons correspond to which operation. These bindings are set in the file mouse-bindings.lisp and may be edited. Currently the following operations can be bound to mouse buttons (the pair following each entry shows the default and the variable that must be changed to modify the default):

- Primary Selection (leftdown, \*prim-select-one-obj\*): The user can either point at a particular object and make it the primary selection, or sweep out a rectangular region of the screen and make all objects that *intersect* the region be primary selections. This operation causes the previous primary selections to be deselected. If the mouse is not pointing at any objects, all primary selections are deselected.



**Figure 1-2:** Different types of selection objects. Squares are for primitive graphical objects, circles are for aggregadgets, and arrows are for objects too small to accommodate grow boxes.

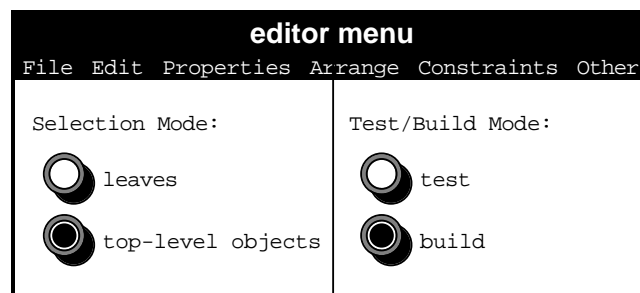
- Secondary Selection (middledown, \*sec-select-one-obj\*): Same as primary selection but makes a secondary selection.
- Add to Primary Selection (shift-leftdown, \*prim-add-to-select\*): Same as Primary Selection except previously selected objects remain selected.
- Add to Secondary Selection (shift-middledown, \*sec-add-to-select\*): Same as add to primary selection but adds to secondary selection.
- Deselect Primary Selection (control-leftdown, \*primary-deselection-button\*): This operation allows the user to deselect primary selections. The user can either point at a specific object or sweep out a rectangular region, in which case all objects that intersect this region will be deselected (if they are primary selections).
- Deselect Secondary Selection (control-middledown, \*secondary-deselection-button\*): Same as Deselect Primary Selection except secondary selections are deselected.
- Primary Select Covered Object (shift-control-leftdown, \*prim-push-sel-under-button\*): This operation allows the user to select covered objects. When the user points at a particular area of the screen, Lapidary determines which object is currently selected, and then deselects it and primary selects the first object that it covers. If no object under the mouse is selected, Lapidary primary selects the top object. If multiple objects under the mouse are selected, Lapidary finds the first unselected object which is under a selected object, selects the unselected object, and deselects the topmost selected object.
- Secondary Select Covered Object (shift-control-middledown, \*sec-push-sel-under-button\*): Same as Primary Select Covered Object except a secondary selection is made.
- Move Object (leftdown, \*move-button\*): This operation allows the user to move an object around the window. The user must point at one of the eight "grow" boxes around the perimeter of box objects, or one of the three "grow" boxes attached to line objects or the arrow if the object is too small to contain grow boxes. If the object is a box object and the user points at one of the corner boxes, the object can move in any direction, if the user points at one of the side boxes, the object can move in only one direction (along the x-axis if the left or right side is chosen and along the y-axis if the top or bottom side is chosen). If the object is a line object, Lapidary will attach the mouse cursor to the point designated by the grow box (either an endpoint of the line or its midpoint) and move the line in any direction. If the object is undersized so that the object does not have grow boxes but instead is pointed at by an arrow, then pointing at the arrow will cause the cursor to be attached to the northwest corner of the object and the object can be moved in any direction.
- Grow Object (middledown, \*grow-button\*): This operation allows the user to resize an

object. The user must point at one of the eight "grow" boxes around the perimeter of the object if the object is a box, one of the endpoint "grow" boxes attached to the object if the object is a line, or the arrow that points at the object if the object is too small to contain the grow boxes. If the object is a box object and the user points at one of the corner boxes, both the object's width and height can change, if the user points at one of the side boxes, only one of the object's dimensions will change (the width if the left or right side is chosen, the height if the top or bottom side is chosen). If the object is a line object, Lapidary will attach the mouse cursor to the point designated by the grow box and move that endpoint while holding the other endpoint fixed. If the object is undersized so that the object does not have grow boxes but instead is pointed at by an arrow, then pointing at the arrow will cause the cursor to be attached to the northwest corner of the object and the object's width and height will both change.

- Object Creation (rightdown, \*obj-creation-button\*): The user sweeps out a region of the screen and Lapidary creates the object selected in the shapes menu.
- Text Editing (rightdown, \*obj-creation-button\*): The user can edit a selected text object by pointing at it and clicking with the object creation button. The user can use any text editing command described in the interactors manual and clicks down on the mouse button to indicate that editing is complete.

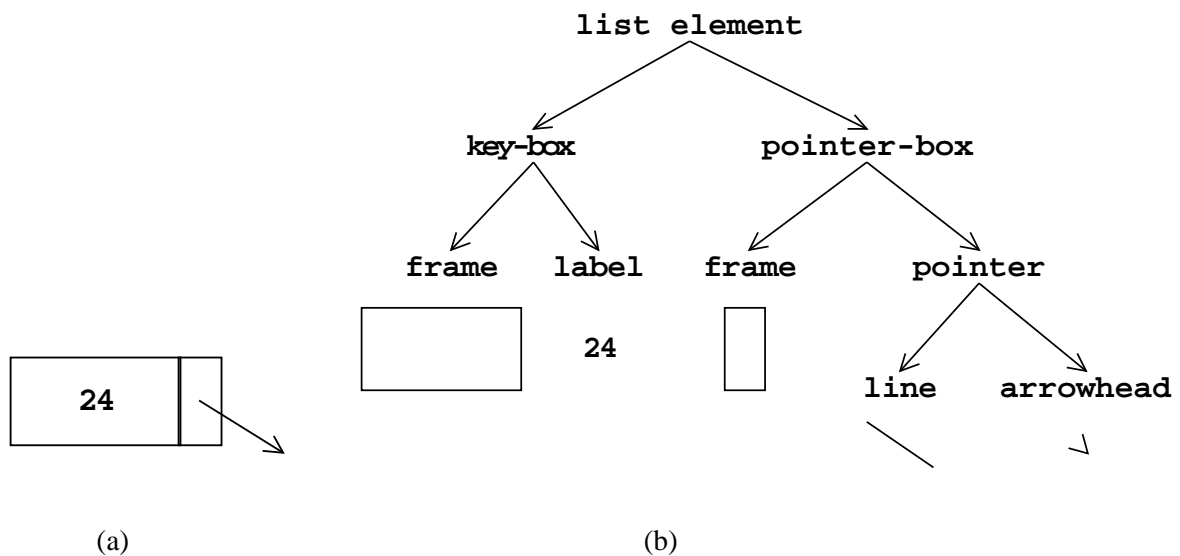
## 1.4. Selection Techniques

A number of features have been added to make it easier to select objects and to perform instancing/duplication on selected objects. The most obvious change is that users can now select either a "leaves" or "top-level objects" selection mode in the editor menu (Figure 1-3). "Leaves" mode causes Lapidary to select leaf elements of an aggregate, while "top-level objects" mode causes Lapidary to select top-level aggregates (objects that do not belong to an aggregate will be selected in either mode). Additional clicks over an object will cause Lapidary to cycle through the aggregate hierarchy as before. For example, when the user clicks on the label shown in Figure 1-4.a, and Lapidary is in "top-level objects" mode, the entire list element is selected (Figure 1-5.a). If the user clicks on the label again, the label is selected (Figure 1-5.b). Clicking once more with the mouse causes the key-box to become selected (Figure 1-5.c). Finally, one more click causes the list element to be selected, at which point the cycle repeats itself. In "leaves" mode, the label would be the first object selected, then the key box, and finally the list element.

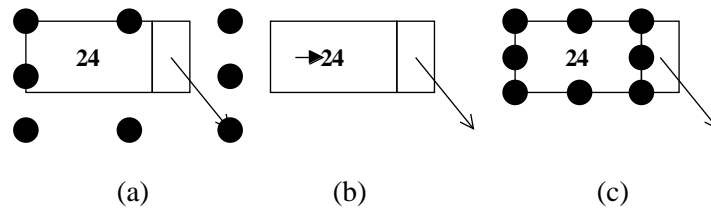


**Figure 1-3:** The user can cause Lapidary to select leaves of aggregates or top-level aggregates by choosing the appropriate selection mode in the editor menu window.

The user can also select the aggregates of covered objects. Previously only leaf elements of covered



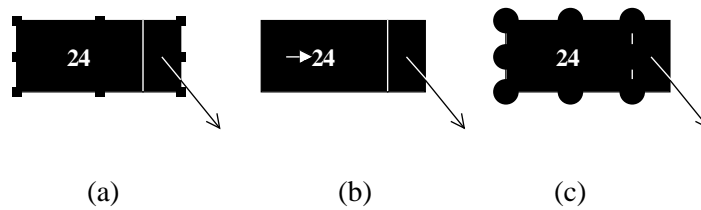
**Figure 1-4:** A list element (a) and the objects used to build this list element (b).



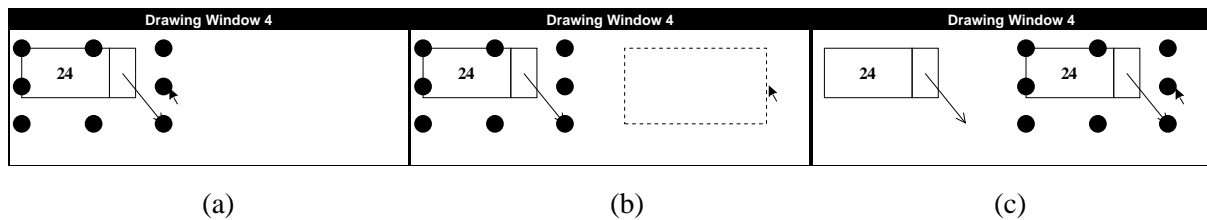
**Figure 1-5:** As the user repeatedly clicks the mouse button over an object, the selection cycles through the aggregate hierarchy shown in Figure 1-4. If Lapidary is in "top-level objects" mode, then the list element is initially selected (a). A second click selects the label (b), and a third click selects the key-box (c).

objects could be selected. As explained in the Lapidary reference manual, covered objects can be selected using either `shift-control-leftdown` for primary selections or `shift-control-middledown` for secondary selections (these bindings can be changed by modifying the `mouse-bindings.lisp` file in the `lapidary` directory). In "top-level objects" mode, the top-level aggregate of a covered aggregate will now be chosen. If the add to selection mouse buttons (`shift-leftdown` for primary selections and `shift-middledown` for secondary selections) are clicked while the mouse cursor is over one of the covered object's selection handles, then the selection will cycle through the aggregate hierarchy (Figure 1-6).

Finally, Lapidary now supports duplicating and instancing operations on the selection handles (as before, the operations can also be performed by selecting them from the pulldown "edit" menu in the editor menu window). If the user clicks the appropriate mouse button while over a selection handle (Figure 1-7.a), either a duplicate (`shift-rightdown`) or instance (`control-rightdown`) of the object will be created and the user can move the object to the appropriate location in the window, or even to another window (Figures 1-7.b and 1-7.c). The selection handle constrains the initial movement of the new object, which facilitates the alignment of objects.



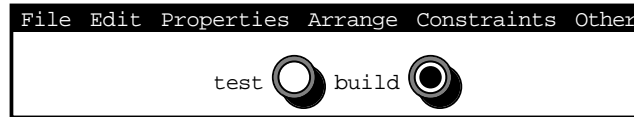
**Figure 1-6:** Selecting a covered object in “leaves” mode. The label is covered by an xor feedback object, so the feedback object is the initial selection (a). Clicking the shift-control-leftdown mouse button pushes the selection down to the covered label (b). Clicking the add to selection button (shift-leftdown over the feedback arrow) causes the selection to cycle up to the next level in the aggregate hierarchy, in this case, the key-box (c).



**Figure 1-7:** Objects can be duplicated or instantiated by clicking on one of the selection handles (a), dragging the new object to the appropriate location (b), and dropping it (c).

## 2. Editor Menu Commands

The commands in Lapidary's pull down menu (Figure 2-1) provide a set of commands for saving and restoring objects, manipulating aggregadgets, applying constraints, and editing properties.



**Figure 2-1:** Lapidary's editor menu

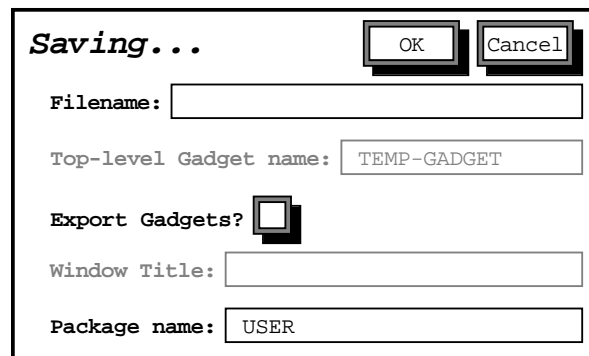
### 2.1. File

- **Save Gadget:** Objects are written out using `opal:write-gadget`, so the file contains a series of create-instance calls. The value in the object's `:known-as` slot is passed as the name parameter to create-instance. For example, if the object's `:known-as` slot is `:white-rect` and the object is a rectangle, the first line of the create-instance would be  

```
(create-instance 'white-rect opal:rectangle)
```

Primary selections are saved before secondary selections, so it is best to make prototypes primary selections and instances of these prototypes secondary selections. The user can also save an entire window by having no objects selected and typing in the string that appears in a window's title bar or icon in the corresponding area of the dialog box.

Lapidary looks at each saved object to determine if the object has any links which Lapidary thinks should be parameters. If Lapidary finds any such links, it pops up the link parameters dialog box and asks the user if these links should be made into parameters (see Section 2.3). Pressing either the OK or CANCEL buttons in the link parameters dialog box allows Lapidary to continue. The CANCEL button in the link parameters dialog box will not cause Lapidary to discontinue the save operation, it will simply cause Lapidary to proceed to the next object.

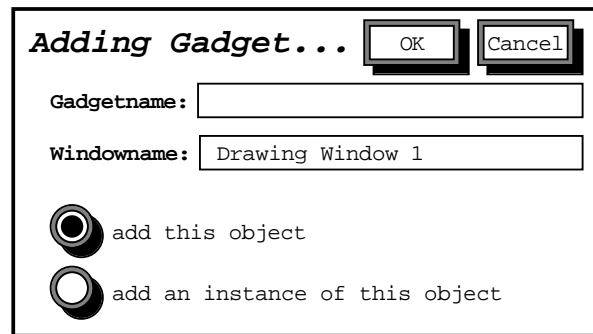


**Figure 2-2:** Save file dialog box

- **Add Gadget:** Users may create objects in the lisp listener and then link them to a Lapidary window. `add-gadget` pops up a dialog box that requests the name of the object to be added and the name of a window to place the object in (Figure 2-3). The name of the object should be the one used in the call to create-instance. For example, the object created by `(create-instance 'my-gadget opal:rectangle)` is named "my-gadget". The name of the window should be the name that appears in the window's title bar or in its icon.

The user has the option of either adding the object itself or an instance of the object to Lapidary. If the user decides to add the object itself and the object has instances, Lapidary





**Figure 2-3:** Add gadget dialog box

will pop up a warning box indicating that editing this object could have unintended consequences on other applications that use this object. For example, it is better to add an instance of a garnet gadgets text button rather than the actual button defined in the gadgets package, since editing the actual button is likely to cause Lapidary to fail (Lapidary uses garnet gadgets text buttons).

- **Quit:** Allows the user to exit Lapidary. It is suggested that before rebooting Lapidary, that the user create a new lisp listener and reload Garnet.

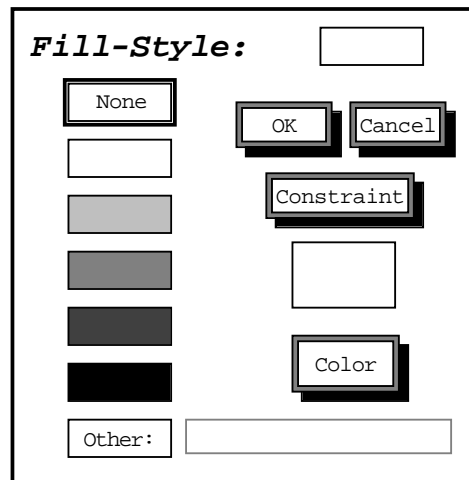
## 2.2. Edit

- **Make Instance:** Creates an instance of the selected object. The selected object is the new object's prototype.
- **Make Copy:** Creates a copy of the selected object. The value of each slot in the selected object will be copied to the new-object. The new object will have the same prototype as the selected object, and thus will inherit from the selected object's prototype rather than the selected object.
- **Delete Object:** Destroys all selected objects.
- **Delete Window:** Pops up a dialog box and asks the user to input the name of a window that appears in a window's title bar or icon. Lapidary then destroys the window.

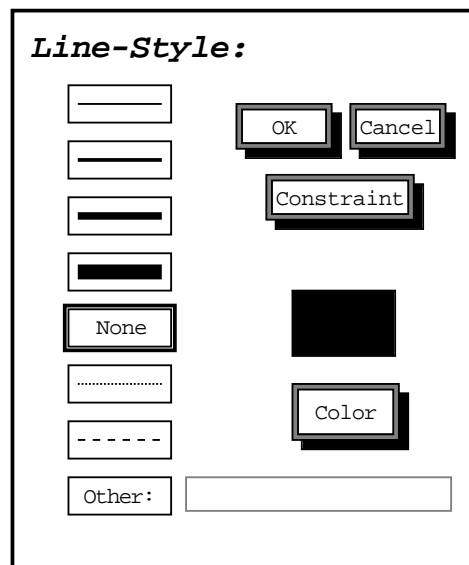
## 2.3. Properties

Lapidary contains four property menus that control an object's line-style, filling-style, draw-function, and font. The line-style and filling-style menus (Figures 2-4 and 2-5) provide a set of commonly used styles, an "Other" option which prompts the user for the name of a style, and a "Constraint" option that allows the user to enter a custom constraint that defines the style (see Section 3 for information on how to enter a custom constraint). The color button pops up a color menu that allows the user to select a pre-defined color or create a new color by mixing hues of red, green, and blue.

- **Filling Style:** Allows the user to set the filling style of selected objects.
- **Line Style:** Allows the user to set the line style of selected objects.
- **Draw Function:** Allows the user to set the draw function of all selected objects. The Opal chapter describes draw functions in more detail.
- **Name Object:** Requests a name from the user (no quotes should be used), converts it to a keyword, and stores it in the :known-as slot of the selected object (if there is more than one selected object, Lapidary will rename the last object the user selected; name object does not distinguish between primary and secondary selections). Lapidary also creates a link with this



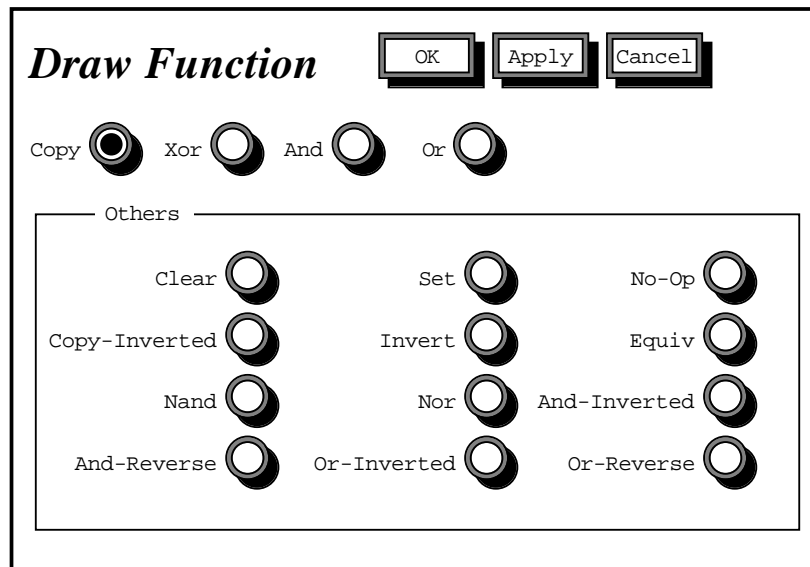
**Figure 2-4:** Filling styles that can be attached to objects in Lapidary



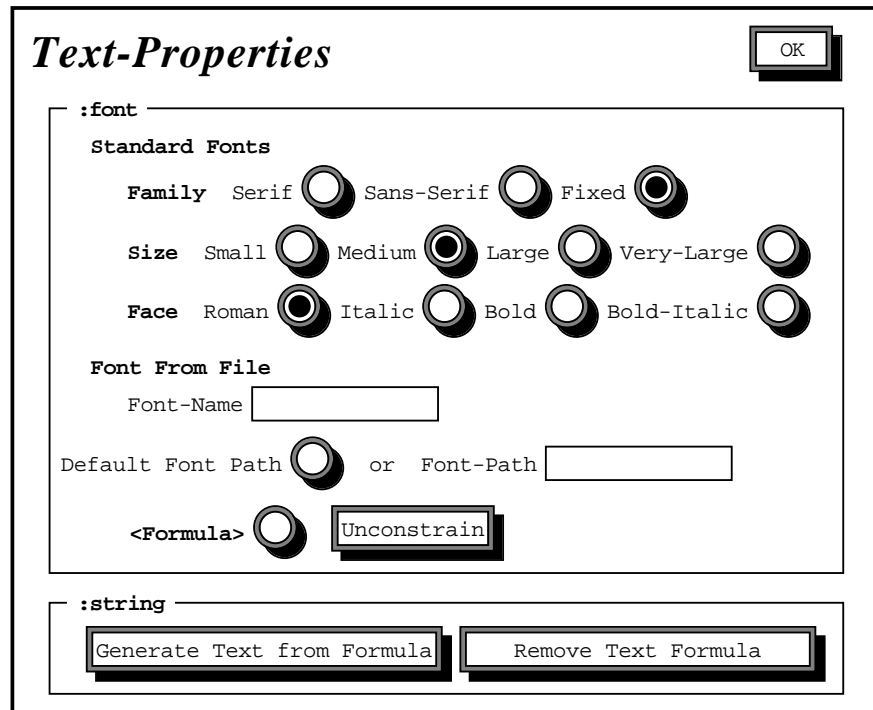
**Figure 2-5:** Line styles that can be attached to objects in Lapidary

name in the object's parent that points to this object. When an object is saved, it will be assigned this name.

- **List Properties:** Brings up a property list for horizontal and vertical lists. This property list allows the user to modify any of the customizable slots of an aggregelist. The list of customizable slots can be found in the Aggrelists chapter.
- **Text Properties:** Allows the user to choose a standard Opal font, to request a font from one of the directories on the user's font path, to request a font from an arbitrary directory, or to enter a custom constraint that determines the font (Figure 2-7). It also allows the user to enter a custom constraint that determines the string of a text object.
- **Parameters:** Allows the user to specify that one or more slots in an object should be parameters (Figure 2-8). A slot that is a parameter will have its value provided at run-time by the application. To create parameters, the user must make both a primary and a secondary selection. The primary selection is the object whose slots are being made into parameters and the secondary selection is the object that the parameters will retrieve their values from. Typically the secondary selection will be the top-level aggregadget that contains the object,



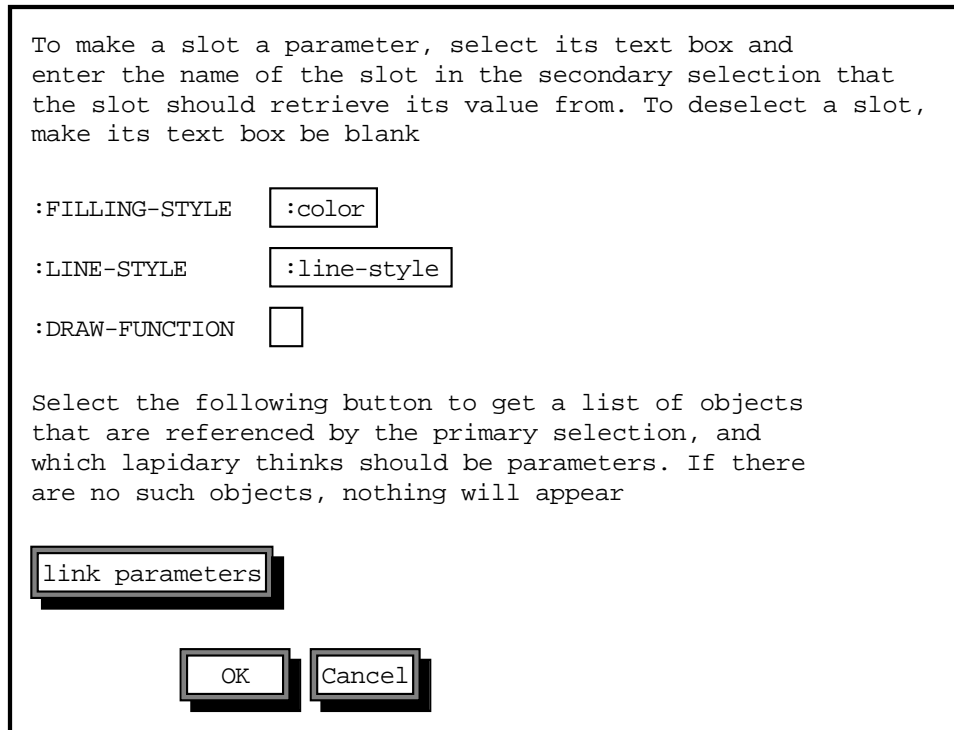
**Figure 2-6:** Draw functions that can be attached to objects in Lapidary



**Figure 2-7:** Lapidary's text properties menu

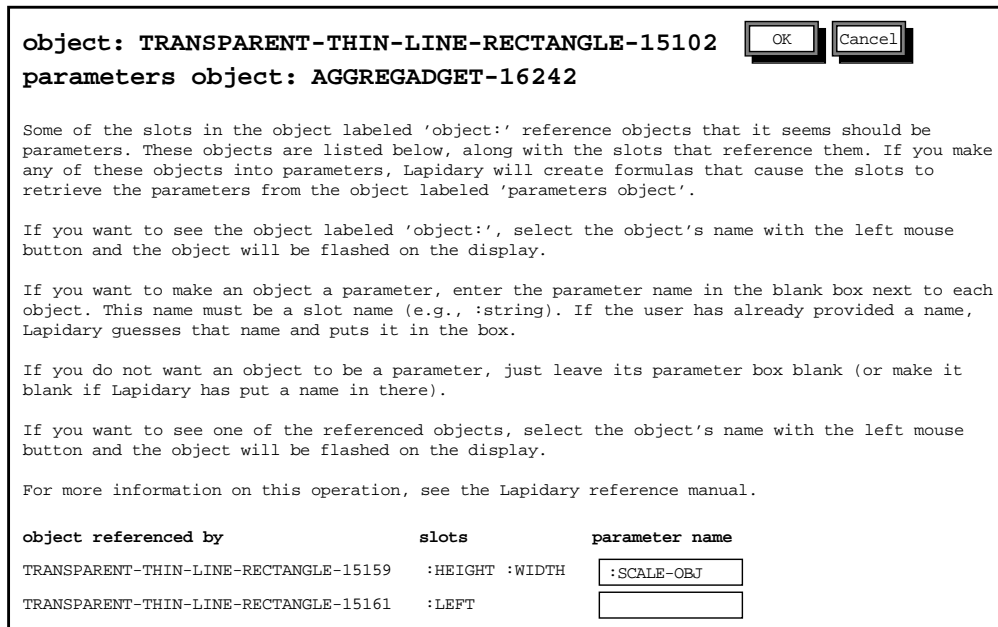
since the top-level aggregadget is the only object that the application should know about (an application should not be required to know the parts of an aggregadget). For example, if a rectangle belongs to an aggregadget, the user might make the rectangle the primary selection and the aggregadget the secondary selection. If the object is already at the top-level, then the object should be both the primary and secondary selection.

To turn a slot into a parameter, select the text box next to the slot and enter the name of the slot in the secondary selection that the slot should retrieve its value from. In Figure 2-8, the rectangle's `:filling-style` slot retrieves its value from the top-level aggregadget's `:color` slot, and the `:line-style` slot retrieves its value from the top-level



**Figure 2-8:** Parameters dialog box

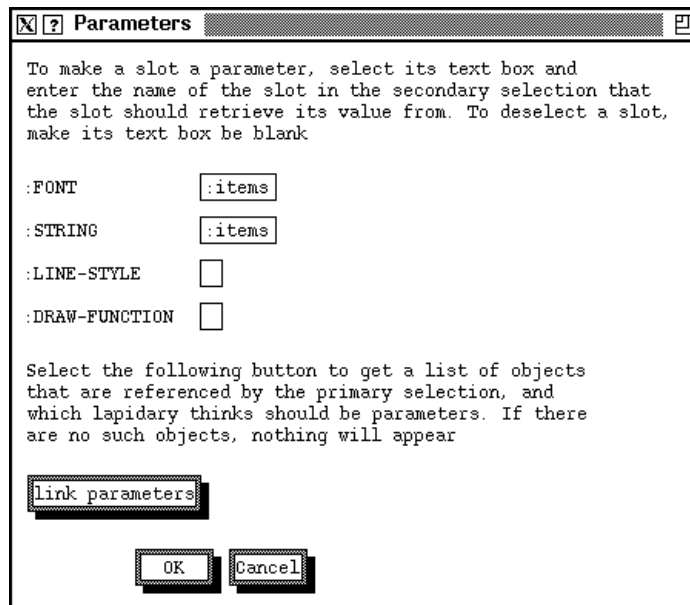
aggregadget's `:line-style` slot. To make the slot no longer be a parameter, make the slot's text box be blank. Lapidary maintains a list of slots for each objects that can be turned into parameters. If the user wants to parameterize a slot that is not displayed in the parameters dialog box, the user can bring up C32 and place a formula in the desired slot that retrieves its value from the top-level aggregadget.



**Figure 2-9:** Link parameters dialog box

The link parameters button in the parameters dialog box allows the user to specify links that

should be parameters. Links are used by Lapidary-generated constraints to indirectly reference other objects. For example, when the user creates a constraint that left aligns one rectangle, say `rect1`, with another rectangle, say `rect2`, Lapidary generates a link in `rect1` that points to `rect2`. When a link references an object that is not part of the primary selection's top-level aggregadget, Lapidary guesses that this link should be a parameter and displays it in the link parameters dialog box (Figure 2-9). For each such link, Lapidary displays the value of the link, the slots that reference the link, and a parameter name, if any, that the user has assigned to this link. The user can change this parameter name by editing it, or can indicate that this link should not be a parameter by making the parameter name blank.



**Figure 2-10:** Parameters dialog box for an Aggrelist

To make a slot depend on an `:items` list in an aggrelist, make any object in the aggrelist be a primary selection and make the aggrelist be the secondary selection. Then enter `:items` in the labeled box for any slot on the parameters menu that should get its value from `:items`. For example, suppose the prototype object for a list is a text object and the string and font slots of the text object should retrieve their values from the aggrelist's `:items` slot. To do this the user makes the aggrelist the secondary selection and one of the text objects in the aggrelist a primary selection. The user then selects the `parameters` option, which causes Lapidary to pop up the parameters menu. Typing `:items` in the type-in fields next to the `font` and `string` slots creates the necessary formulas that link these slots to the `:items` slot in the aggrelist (Figure 2-10). The `:items` slot of the aggrelist will now contain a list of the form `((string1 font1) (string2 font2)...(stringN fontN))`.

If the prototype object is an aggregadget (such as a labeled box that contains a rectangle and a piece of text), then any of the parts of the aggregadget, and the aggregadget itself, can have slots that depend on the aggrelist's `:items` slot. This is done by parameterizing the parts one at a time. For example, if the string slot of the text object and the filling-style slot of the rectangle should be parameters, the user could first select the rectangle and parameterize it, then select the text object and parameterize it. Lapidary does not follow any easily described rules in constructing the `:items` list (e.g., the string and font values could easily have been reversed in the above list), so users should look at the `:items` list Lapidary constructs before writing their own.

If a slotname besides `:items` (e.g., `:string`) is entered in a type-in field, then the slot is treated as an ordinary parameter, and all items in the list will have a formula that accesses this slot in

the `aggrelist`. For example, if a list consists of rectangles, and the rectangles should all have the line-style that is passed to the `aggrelist`, then the user would select one of the rectangles and enter the an appropriate name, such as `:line-style`, next to the `:line-style` slot of the rectangle.

## 2.4. Arrange

- **Bring to Front:** Brings the selected objects to the front of their `aggregadget` (i.e., they will cover all other objects in their `aggregadget`). If multiple objects are selected, it brings the objects to the front in their current order.
- **Send to Back:** Sends the selected objects to the back of their `aggregadget` (i.e., they will be covered by all other objects in their `aggregadget`). If multiple objects are selected, it sends the objects to the back in their current order.
- **Make Aggregadget:** Creates a new `aggregadget` and adds all selected objects (both primary and secondary selections) to it. The selected objects must initially belong to the same `aggregadget` or else Lapidary will print an error message and abort the operation. The `:left` and `:top` slots of the objects added to the `aggregadget` are constrained to the `aggregadget` unless they were already constrained (if the object is a line, the `:x1`, `:y1`, `:x2`, and `:y2` slots are constrained). The constraints tie the objects to the northwest corner of the `aggregadget` and use absolute offsets based on the current position of the objects. Thus if an object is 10 pixels from the left side of the `aggregadget` (the bounding box of the `aggregadget` is computed from the initial bounding boxes of the objects), the object's `:left` slot will be constrained to be 10 pixels from the left side of the `aggregadget`. If the object is a line, the object's endpoints will be tied to the `aggregadget`'s northwest corner by absolute fixed offsets. These constraints cause the objects to move with the `aggregadget` when the `aggregadgets` moves. If the user wants different constraints to apply, the user can primary select an object, secondary select the `aggregadget`, and attach a different constraint. The `aggregadget` derives its width and height from its children, so the `:width` and `:height` slots of the children are not constrained to the `aggregadget`. Because the `aggregadget` computes its width and height from its children, it is not permitted to resize an `aggregadget`.
- **Ungroup:** Destroys selected `aggregadgets` and moves their components to the `aggregadgets`' parents.

## 2.5. Constraints

- **Line Constraints:** Brings up the line constraints dialog box (Figure 3-2).
- **Box Constraints:** Brings up the box constraints dialog box (Figure 3-1)
- **C32:** Brings up C32. Each primary and secondary selection is displayed in the spreadsheet, and additional Lapidary objects can be displayed using the `Point to Object` command. While Lapidary is running, only objects in Lapidary's drawing windows can be displayed in the spreadsheet. Nothing will happen if the user attempts to execute the `Point to Object` command on an object which is not in a Lapidary drawing window. The C32 chapter describes how to use C32 and Section 3.3 describes a number of modifications Lapidary makes to C32.

## 2.6. Other

- **Interactors:** Displays a menu of interactors that the user can choose to look at. Once the user selects an interactor, the information from that interactor will be displayed in the appropriate interactor dialog box (see Section 4) and the user is free to change it. In addition,

menu items are provided for the five Garnet-defined interactor types: choice (encompassing both menu and button interactors), move/grow, two-point, text, and angle. If the user has selected a set of objects, then the interactors menu will contain all interactors associated with these objects. Lapidary will display all interactors whose `:start-where` slot references these objects, or whose `:feedback-obj` or `:final-feedback-obj` points at these objects. If no objects are selected, then the interactors menu will contain all interactors that have been created in Lapidary.

- **Clear Workspace:** Deletes all objects from Lapidary but does not destroy any of the drawing windows.

## 2.7. Test and Build Radio Buttons

- **Test:** Deactivates the Lapidary interactors that operate on the drawing windows and activates all user-defined interactors. This allows the user to experiment with the look-and-feel that the user has created.
- **Build:** Deactivates all user-defined interactors and reactivates the Lapidary interactors, allowing the user to modify the look-and-feel.

## 3. Creating Constraints

Lapidary provides two menus for creating constraints, one that deals with "box" constraints (constraints on non-line objects) and one that deals with line constraints. In addition, several of the property menus provide a custom constraint option that allows the user to input a constraint that determines the property. Each of the menus contains buttons labeled with tiny rectangular boxes that indicate how an object will be positioned if the constraint associated with that button is chosen. The rectangular boxes in the buttons are colored black to indicate that the primary object is the object that will be constrained, and the white rectangular boxes positioned at the four corners of the rectangle in the box constraint menu indicate that the secondary selection is the object that will be referenced in the constraint.

The Box and Line Constraint dialog boxes, can be used separately from Lapidary (see section 3.4).

The constraint menus can display the current position and size of a primary selection. By pressing the `Show Constraints` button in the constraint menus, the user can see what types of constraints are on the slots of an object. If two objects are selected, Lapidary will display the types of the constraints between the two objects.

Finally, both box-like and line-like objects can be constrained to the center of a line by selecting the new centering button in the `obj-to-reference` section of the Line Constraint dialog box.

### 3.1. Box Constraints

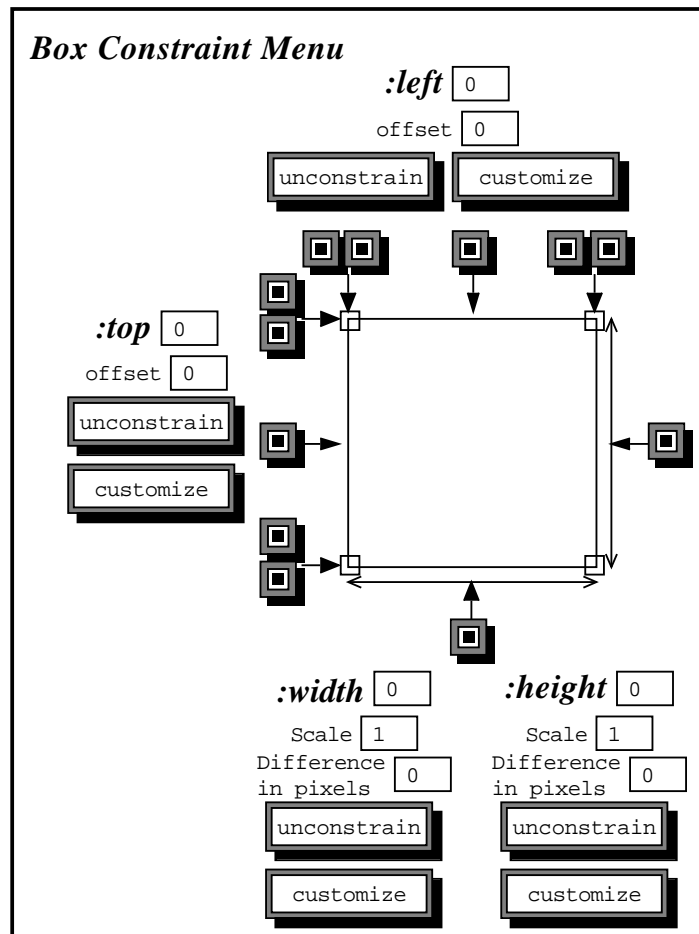
The box constraint menu allows constraints to be attached to the `:left`, `:top`, `:width`, and `:height` of an object (see Figure 3-1). The user attaches constraints by first selecting the object to be constrained (a primary selection) and the object to be referenced in the constraint (a secondary selection). The user then selects the appropriate buttons in the box constraint menu. The possible constraints for the `:left` slot are:

- `left-outside`: The right side of the primary selection is aligned with the left side of the secondary selection.
- `left-inside`: The left side of the primary selection is aligned with the left side of the secondary selection.
- `center`: The center of the primary selection is aligned with the center of the secondary selection.
- `right-inside`: The right side of the primary selection is aligned with the right side of the secondary selection.
- `right-outside`: The left side of the primary selection is aligned with the right side of the secondary selection.

The possible constraints for the `:top` slot are:

- `top-outside`: The bottom side of the primary selection is aligned with the top side of the secondary selection.
- `top-inside`: The top side of the primary selection is aligned with the top side of the secondary selection.
- `center`: The center of the primary selection is aligned with the center of the secondary selection.
- `bottom-inside`: The bottom side of the primary selection is aligned with the bottom side of the secondary selection.
- `bottom-outside`: The top side of the primary selection is aligned with the bottom side of the





**Figure 3-1:** Lapidary menu for attaching constraints to box objects

secondary selection.

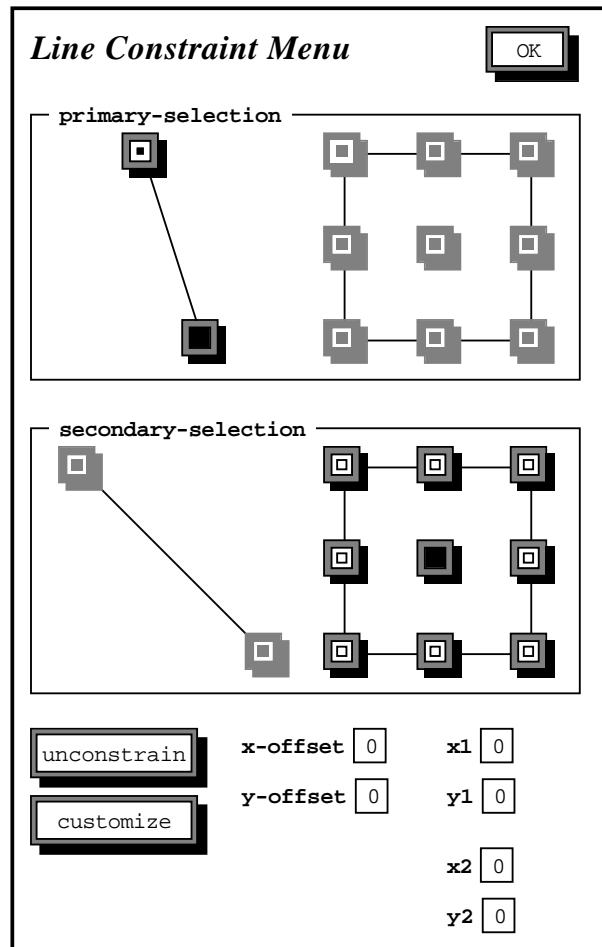
The only option for the `:width` slot is to constrain the width of the primary selection to the width of the secondary selection and the only option for the `:height` slot is to constrain the height of the primary selection to the height of the secondary selection. In addition, each of the four slots may have a custom constraint attached to them (see Section 3.3). Each of the four slots also has an "Unconstrain" option that destroys the constraint attached to that slot.

The constraints in the box constraint menu can be fine-tuned by entering offsets for the constraints, and in the case of the size slots (width and height), scale factors as well. When an object is centered with respect to another object, the offset field changes to a percent field denoting an interval where 0% causes the center point of the constrained object to be attached to the left or top side of the object referenced in the constraint and 100% causes the center point of the constrained object to be attached to the right or bottom side of the object referenced in the constraint. By default this percentage is 50. The *Difference in pixels* and *Scale* factors cause the width and height constraints to be computed as  $Scale * Dimension + Difference\ in\ pixels$ .

Finally, each of the slots has a labeled box next to its name that allows the user to type in an integer that will be placed in that slot. If there is already a constraint in the slot, the constraint will not be destroyed so the value will only temporarily override the value computed by the constraint (the next time the constraint is recomputed, the value will be lost). This operation works only when there is one primary selection and no secondary selections.

### 3.2. Line Constraints

The line constraint menu allows the endpoints of a line to be attached to other objects or the `:left` and `:top` slots of a box object to be constrained to the endpoint of a line (Figure 3-2). The buttons on the box and line object in Figure 3-2 indicate the various locations where the endpoint of a line can be attached to a box or line object or where a point of a box can be attached to a line. Thus the two endpoints of a line can be attached to any of the corners, sides, or center of a box object and any of the corners, sides, or center of a box object can be attached to the endpoints of a line.



**Figure 3-2:** Lapidary menu for attaching constraints to line objects

The line object in the constraint menu is oriented in the same direction as the selected line in the drawing window, so that the user knows which endpoint is being constrained. The buttons with the blackened rectangles indicate the points that can be constrained in the primary selection. Similarly, the buttons with the white rectangles indicate the points in the secondary selection that the primary selection can be attached to.

The `Unconstrain` button at the bottom of the menu allows the user to destroy the constraint on the selected point and the `Customize` button allows the user to input a custom constraint (described in Section 3.3). Finally the `x-offset` and `y-offset` labeled boxes allow the user to enter offsets for the constraint. All offsets are added to the value computed by the constraint. For example, an `x-offset` of 10 causes an endpoint constrained to the northwest corner of a box object to appear 10 pixels to the right of that corner.

The end points of a line can be set directly by typing in values for `x1`, `y1`, `x2` and `y2`. This function is only

active if there is a single line as the primary selection and no secondary selections.

### 3.3. Custom Constraints

When the user selects the custom constraint option in any of the constraint or property menus, Lapidary brings up the C32 spreadsheet and a formula window for the desired slot. The user should enter a formula and press OK (or cancel to stop the operation). Both the OK and cancel buttons in the formula window will make C32 disappear.

Information on C32 can be found in the C32 chapter. However, Lapidary modifies C32 in a number of ways that are important for a user to know. First, it generates indirect references to objects rather than direct references. A direct reference explicitly lists an object in a constraint, whereas an indirect reference accesses the object indirectly through a link. For example, `(gv rect1 :left)` is a direct reference to `rect1`, whereas `(gv1 :link0 :left)` is an indirect reference to `rect1` (this assumes that a pointer to `rect1` is stored in `:link0`). If the user always generates references using the C32 functions `Insert Ref from Mouse` and `Insert Ref from Spread`, then Lapidary will automatically generate indirect references and create appropriate link names. The user can edit these link names by bringing up the parameters menu and hitting the link parameters button (see Section 2.3). However, if the user inserts the references by typing them in, then the user should take care to use the `gv1` form and create the appropriate links. When the formula is completed, Lapidary checks whether there are any direct references in the formula and generates a warning if there are. At this point the user has the option of editing the formula or continuing with the formula as is. If the user chooses to leave direct references in the formula, Lapidary may not be able to generalize it, so the formula may behave strangely if it is inherited.

The second change Lapidary makes is in copying formulas. Lapidary copies all the links that the formula references to the object which is receiving the copied formula. If the links should point to new objects, the user must manually change them by selecting the `Show All Slots` option in C32 and editing the appropriate links (the names of the links that need to be modified can be found by looking at the formula).

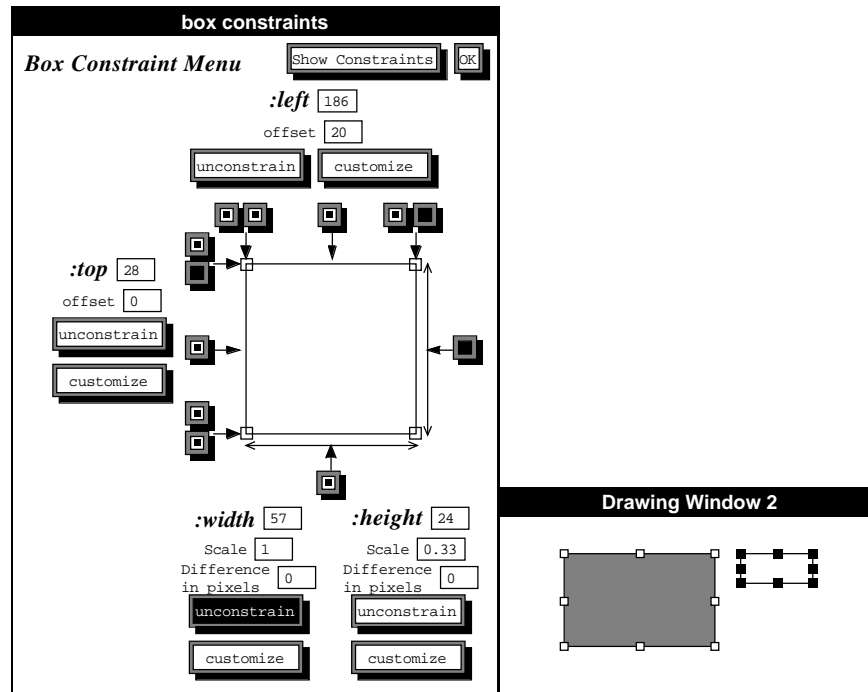
### 3.4. The Constraint Gadget

The constraint gadget allows a set of standard constraints to be attached to various Garnet objects. The constraint gadget provides two menus: a box constraint menu for specifying box-type constraints (Figure 3-3) and a line constraint menu for specifying line-type constraints (Figure 3-4). The module can be loaded independently of Lapidary with `(garnet-load "lapidary:constraint-gadget-loader")`.

#### 3.4.1. Box Constraint Menu

The box constraint menu allows the `:left`, `:top`, `:width`, and `:height` of objects to be constrained. Constraints on the left slot may include:

- Left-Outside - attach the right side of the constrained object to the left side of the reference object;
- Left-Inside - align the left side of the constrained object to the left side of the reference object (this is left alignment);
- Center - center the constrained object with respect to the reference object's center;
- Right-Inside - attach the right side of the constrained object to the right side of the reference object (this is right alignment); and
- Right-Outside - attach the left side of the constrained object to the right side of the reference object.



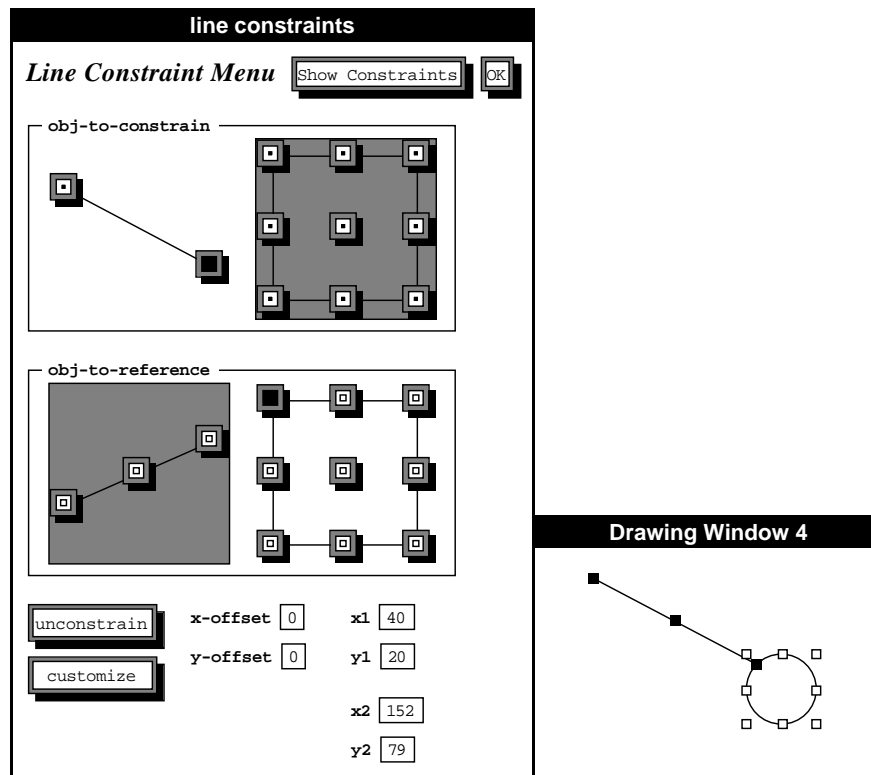
**Figure 3-3:** The constraint gadget's menu for box-like objects on the left, and a drawing window on the right. The white rectangle in the drawing window is the object to be constrained and the gray rectangle is the object to be referenced in the constraint. The white rectangle is constrained to be offset from the right of the gray rectangle by 20 pixels, and aligned at the top-inside of the gray rectangle. The white rectangle's width is not constrained and it is 33% as tall as the gray rectangle. If the gray rectangle changes, the white one will be adjusted automatically.

Constraints on the top slot may include:

- Top-Outside - attach the bottom side of the constrained object to the top side of the reference object;
- Top-Inside - align the top side of the constrained object to the top side of the reference object (this is top alignment);
- Center - center the constrained object with respect to the reference object's center;
- Bottom-Inside - attach the bottom side of the constrained object to the bottom side of the reference object (this is bottom alignment); and
- Bottom-Outside - attach the top side of the constrained object to the bottom side of the reference object.

An object may be constrained to another object's width or height using the width and height portions of the box constraint menu.

The offset boxes for the left and top slots, and the scale and difference boxes for the width and height slots provide ways of fine-tuning the constraints. The offset boxes for the left and top slots allow the user to add a constant pixel offset to a constraint. For example, the value of 20 in the left offset box in Figure 3-3, causes the white rectangle to be placed 20 pixels to the right of the gray rectangle. When the user



**Figure 3-4:** The constraint-gadget menu for line-like objects on the left, and a drawing window on the right. The arrow in the drawing window is the object to be constrained and the circle is the object to be referenced in the constraint. In the “obj-to-constrain” section of the constraint menu, the line feedback object has been rotated so that it has the same orientation as the selected arrow, and the box feedback object has been disabled (grayed-out). In the “obj-to-reference” section the line feedback object has been disabled since the object to be referenced in the constraint is a box-like object. The darkened buttons on the right endpoint of the line feedback object and the left corner of the box feedback object indicate that the right endpoint of the arrow is attached to the left corner of the circle.

presses a center button, the label for the offset box turns to percent, since the user enters a percentage for the center (50% is the default). In this case the constraint has the form “reference-obj.left + (percent \* reference-obj.width) - constrained-obj.width / 2”. The scale and difference boxes for the width and height slots allow the user to create constraints of the form “scale \* {width, height} + difference”.

Pressing an unconstrain button will remove a constraint from the appropriate slot of the constrained object, if there is one. Pressing a customize button will bring up C32 with a formula box for the appropriate slot. More information on using C32 with the constraint gadget is included in Section 3.4.4.

The boxes next to the slot names allow the user to enter specific values for an object’s left, top, width, and height. This option is useful when the designer wants specific values for these slots, rather than the approximate values obtained using direct manipulation. When a value is entered in one of these boxes, the constraint in that slot is destroyed if one exists. When an object is selected, these boxes reflect the values of the object’s bounding box.

### 3.4.2. Line Constraint Menu

The line constraint menu supports three types of constraints: line to line, line to box, and box to line. To make it clear what type of constraint is to be applied, the line constraint menu displays the types of the object to be constrained and the object to be referenced in the constraint (either a line or a box). It does so by displaying two sections, an `obj-to-constrain` section and an `obj-to-reference` section. Each section displays both a line and a box. The line objects have buttons indicating that endpoints can be constrained and the box objects have buttons indicating that edges, corners, and the center can be constrained. The line in the `obj-to-reference` section has an additional button in its center, indicating that objects can be constrained to the center of a line (the line in the `obj-to-constrain` menu does not have a center button, but the user can still constrain the center of a line by pressing the `customize` button in the line constraint menu). When an object is selected, the line constraint menu grays out whichever feedback object does not correspond to the type of the selected object. In addition, if the selected object is a line, then the appropriate feedback line is rotated, so that its alignment matches the alignment of the selected line. This rotation makes it clear which endpoint of the selected line is being constrained, or being used as a reference in a constraint.

The line constraint menu allows either endpoint of a line to be constrained, and any corner, side, or center of a box object to be constrained. To create a constraint, the designer pushes the appropriate button on the enabled feedback objects in both the `obj-to-constrain` and the `obj-to-reference` sections. The connection between two points can be fine-tuned using the `x-offset` and `y-offset` fields. For example, if the designer wanted a line to be connected to the left side of a box, 10 pixels from the top, the designer could constrain the line to the left corner of the box, and enter a `y-offset` of 10.

The `x1`, `y1`, `x2`, and `y2` boxes allow the designer to specify an exact pixel position for the endpoints of a line. When an object is selected, these boxes are updated to reflect the values of the line's endpoints.

The `unconstrain` button destroys the constraints on the endpoint of a line or the position of a box. If a line is selected, then a button in the `obj-to-constrain` section denoting the appropriate endpoint should be selected. If neither endpoint button is selected, the constraint gadget tries to figure out which endpoint to unconstrain by checking if only one endpoint is constrained. If only one endpoint is constrained, it unconstrains this endpoint. Otherwise, if both endpoints are constrained, the constraint gadget asks the user to select one of the endpoint buttons in the `obj-to-constrain` section of the line constraint menu.

The `customize` button brings up C32. Since it is unclear whether the user will constrain both endpoints of a line, or just one of the `x` and `y` coordinates, the constraint gadget does not bring up formula boxes for the slots.

### 3.4.3. Programming Interface

The constraint gadget can be created (or made visible, if already created) by executing one of the `-do-go` or `show-` functions described in section 3.4.3.2. Certain slots of the gadget, described in section 3.4.3.1, are then set with the objects to be constrained. When the user operates the buttons in the dialog box, constraints will be set up among the indicated objects.

#### 3.4.3.1. Slots of the Constraint Gadget

The constraint gadget exports one object called `gg::*constraint-gadget*`. This object contains four settable slots:

- `:obj-to-constrain` - The object which should be constrained. This slot expects only one object — it will not take a list.
- `:obj-to-reference` - The object which should be referenced in the constraint. This slot expects only one object — it will not take a list.

- `:top-level-aggregate` - The top level aggregate containing constrainable objects. If the aggregate associated with a window is the top level aggregate, this slot may be left NIL (the default). However, if, for example, the window contains an editor aggregate and a feedback aggregate, then the `:top-level-aggregate` slot should be set to the editor aggregate.
- `:custom-function` - A function that is executed whenever a constraint is attached to a slot. The function should take three parameters: an object, a slot, and a formula. The function is called after the formula has been installed on the slot, but before the formula has been evaluated. *This function is not called when the user calls the `c32` function and provides a `c32-custom-function` as a parameter (see Section 3.4.3.2 for details on this function and its parameters. The function is not called in this case since the constraint gadget does not install the formula if the `c32-custom-function` is provided.)*

It is also possible to prevent either the box-constraint or line-constraint menus from attaching a constraint to a slot by adding the slot's name to a list in the `:do-not-alter-slots` of an object. For example, to prevent a constraint from being attached to the `:width` or `:height` slots of a text object, the list `('(:width :height))` could be placed in the object's `:do-not-alter-slots` slot. If the user tries to attach a constraint to that slot, an error box will be popped up indicating that a constraint cannot be attached to that slot. C32 does not recognize the `:do-not-alter-slots`, and therefore the box-constraint and line-constraint menus cannot prevent the user from inserting a formula into a forbidden slot if a customize button is chosen.

### 3.4.3.2. Exported Functions

The following functions are exported from the constraint gadget module:

`lapidary:Box-Constraint-Do-Go` [Function]

`lapidary:Line-Constraint-Do-Go` [Function]

These functions create the Box and Line Constraint dialog boxes. They should not be executed multiple times, since there is only one `constraint-gadget` object. If the user clicks on an "OK" button and makes the dialog boxes invisible, then the following functions can be called to make them visible again:

`lapidary:Show-Box-Constraint-Menu` [Function]

`lapidary:Show-Line-Constraint-Menu` [Function]

These functions make the Box and Line Constraint dialog boxes visible. They can only be called after the `-do-go` functions above have been called to create the dialog boxes.

`gg:c32 &optional object slot` [Function]  
`&key left top c32-custom-function prompt`

This function causes `c32` to come up, with the *object* displayed in the first panel of the `c32` window. The formula in *slot* will be displayed in `c32`'s formula editing dialog box. The keyword parameters are as follows:

*left, top* - Controls placement of query box that users use to indicate that they are done with `C32`.

*c32-custom-function* - A function to be executed when the user hits the OK button in a formula window in `C32`. The function should take three parameters: an object, a slot, and a formula. If a custom function is provided, the formula will not be installed in the slot (thus the function in `:custom-function` will not be called—it must be called explicitly by the *c32-custom-function* if it should be executed). This gives the *c32-custom-function* an opportunity to defer the installation of the formula. For example, in Lapidary, the user can create formulas that define the values of various slots in an interactor, but until the user presses the “create-interactor” or “modify” buttons, the formulas should not be installed. Thus the Lapidary *c32-custom-function* places the formulas on a queue, but does not install them.

The constraint gadget stores the links that this formula uses in a meta-slot in the formula called `:links`. Like the formula, the links are not installed. That is, the link slots do not exist (unless another formula already uses them). Because the links have not been installed, the constraint gadget stores the links and the objects they point to in another meta-slot in the formula called `:links-and-objs`. The contents of this slot have the form `(list (cons (link-name object))*)`. Links that already exist because another formula uses them will not be on this list.

The *c32-custom-function* or the application can install the links by calling the c32 function `install-links` which takes a formula and the object that the links should be installed in as arguments (the object that is passed to *c32-custom-function* is the object that should be passed to `install-links`). `install-links` will create the links, and if the link points to an object that is in the same aggregate as the object containing the link, `install-links` will create a path to the reference object and store it in the link slot. `install-links` destroys the `:links-and-objs` slot, so the *c32-custom-function* or application should take care to save the contents of this slot if they need to make further use of this information.

*cg-destroy-constraint* - Destroys a constraint created by the constraint gadget. Required parameters are an object and a slot.

*valid-integer-p* - Determines if a string input by a Garnet gadget contains a valid integer. If it does not, the gadget's original value is restored and an error message is printed. Required parameters are a gadget and a string.

`c32::Install-Links formula obj`

[Function]

This function is provided by c32, though it is not exported. As mentioned above, it is useful for installing links when a custom function is provided in c32. The *formula* should have a `:links-and-objs` slot, whose value should be a list of the form `((link-slot-name object) (link-slot-name object) ...)`. The *object* parameter names the object which the links should be installed in.

### 3.4.3.3. Programming with Links

Each constraint contains indirect references to objects rather than direct references. The set of links it uses to make these indirect references is contained in the `:links` meta-slot of the formula and the name of the offset slot it uses is contained in the `:offset` meta-slot. If the formula involves the width or height slots, there is also a `:scale` meta-slot, containing the name of the scale slot that the formula uses. The constraint gadget generates link and offset names by appending the suffixes `-over` and `-offset` to the name of the slot that is being constrained. For example, if the left slot is being constrained, the link name will be `:left-over` and the offset name will be `:left-offset`. These slot names are stored in a formula's `:links` and `:offset` meta-slots. For width and height slots, scale names are generated by appending the suffix `-scale` to the slot name. Thus the scale slot for a height constraint would be named `:height-scale`. When C32 generates link names, it generates them by appending a number to the prefix `link-`. Thus it generates links such as `:link-0` and `:link-1`.

### 3.4.4. Custom Constraints

When the user selects the custom constraint option in any of the constraint menus, the constraint gadget brings up the C32 spreadsheet and a formula window for the desired slot. The user should enter a formula and press OK (or cancel to stop the operation). Both the OK and cancel buttons in the formula window will make C32 disappear.

The constraint gadget modifies C32 in a number of ways that are important for a user to know. First, it generates indirect references to objects rather than direct references. A direct reference explicitly lists an object in a constraint, whereas an indirect reference accesses the object indirectly through a link. For example, `(gv RECT1 :left)` is a direct reference to `RECT1`, whereas `(gv1 :link0 :left)` is an



indirect reference to `RECT1` (this assumes that a pointer to `RECT1` is stored in `:link0`). If the user always generates references using the C32 functions `Insert Ref from Mouse` and `Insert Ref from Spread`, then the constraint gadget will automatically generate indirect references and create appropriate link names. The user can edit these link names by finding them in the spreadsheet and modifying them. However, if the user inserts the references by typing them in, then the user should take care to use the `gvl` form and create the appropriate links. When the formula is completed, the constraint gadget checks whether there are any direct references in the formula and generates a warning if there are. At this point the user has the option of editing the formula or continuing with the formula as is. If the user chooses to leave direct references in the formula, the constraint gadget may not be able to generalize it, so the formula may behave strangely if it is inherited.

The second change the constraint gadget makes is in copying formulas. The constraint gadget copies all the links that the formula references to the object which is receiving the copied formula. If the links should point to new objects, the user must manually change them by selecting the `Show All Slots` option in C32 and editing the appropriate links (the names of the links that need to be modified can be found by looking at the formula).

### 3.4.5. Feedback

The user can determine which constraints are attached to an object by selecting the object and an optional second object that the object may be constrained to, and then selecting the `Show Constraints` option. The appropriate constraint buttons will be highlighted and the offset fields set to the correct values. If only one object is selected, then all constraints that the constraint menu can represent will be shown. For example, the box constraint menu would display the constraints on the left, top, width, and height slots. If there are two selections, a constrained object and a reference object, then only the constraints in the constrained object that depend on the reference object are shown.

## 4. Interactors

Lapidary provides a set of dialog boxes that allow a user to define new interactors or modify existing ones. To create or modify an interactor, select the `Interactors` command from the Lapidary editor menu. Lapidary will display a menu listing Garnet-defined and user-defined interactors that may be viewed. Select the desired interactor and Lapidary will display the appropriate interactor dialog box.

All interactor dialog boxes have a number of standard items, including a set of action buttons, a name box, a `:start-where` field, and buttons for events. In addition, each dialog box allows the user to set the most commonly changed slots associated with that interactor. Other slots may be set using C32 (see section 3.3).

The name field allows the user to type in a name for the interactor. The name is not used to name the interactor, but instead is converted to a keyword and stored in the interactor's `:known-as` slot. If the interactor is saved, the user-provided name will be placed in the name parameter field for `create-instance`.

### 4.1. Action Buttons

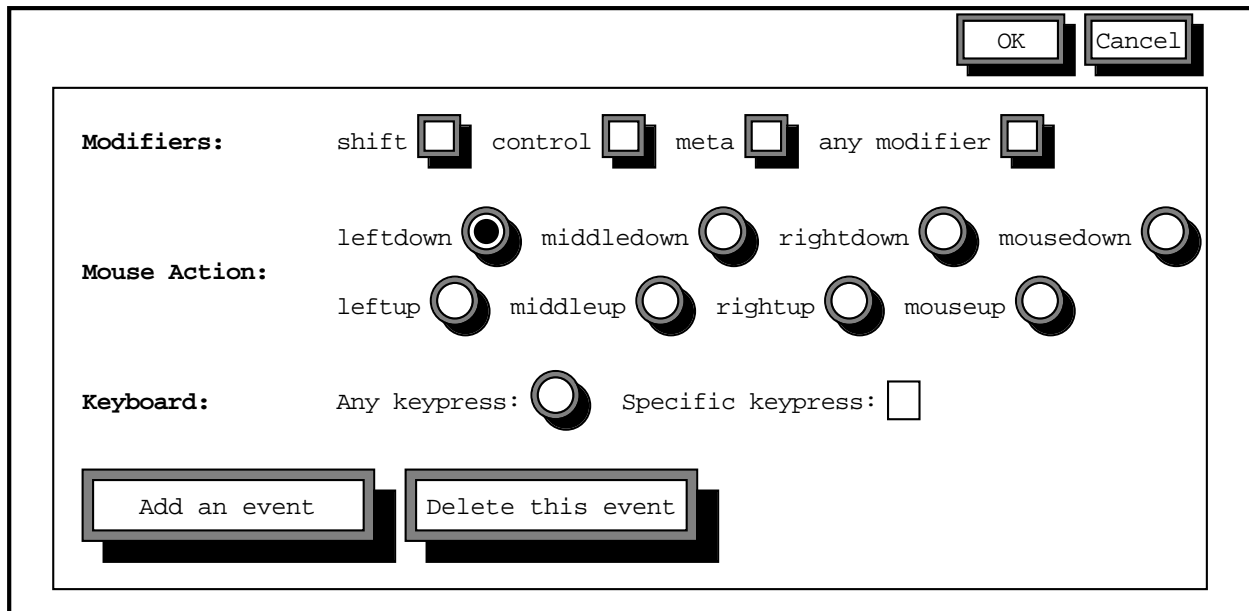
The action buttons permit the following types of operations:

- **Create Instance:** This operation creates an instance of the displayed interactor and, if the user has modified any of the slot values, overrides the values inherited from this interactor with the modified values. In addition, Lapidary examines the `:start-where` field of the new interactor and if the start-where includes an aggregadget, adds the interactor to the aggregadget's behavior slot.
- **Modify:** This operation stores any changes that the user has made to the interactor's slots in the interactor.
- **Destroy:** This operation destroys the interactor.
- **Save:** This operation prompts the user for a file name and then writes out the interactor.
- **C32:** This operation brings up C32 and displays the interactor in the spreadsheet window. The user can then edit any slot in the interactor. Any changes the user makes will not be discarded by the `Cancel` button. It is generally advisable to bring up the C32 menu only *after* the interactor has been created. (the one exception to this rule is when C32 appears as the result of pressing a formula button. If the user enters a formula in the formula window, the formula will be installed in the instance). Otherwise the user will end up editing the prototype for the interactor to be created, instead of the interactor itself. The C32 chapter describes how to operate C32 and Section 3.3 describes the modifications Lapidary makes to C32.
- **Cancel:** This operation discards any changes the user has made to the dialog box since the last `create-instance` or `modify` command.

### 4.2. Events

Lapidary allows the user to define the start, stop, and abort events of an interactor using event cards. Each card defines one event and a list of events can be generated from a deck of cards. Each interactor dialog box contains buttons that pop up a window for each event that defines a start, stop or abort event. A sample event card is shown in Figure 4-1. Selecting `Delete this event` will cause this event to be deleted. However, Lapidary will not allow you to delete an event card if it is the only one that exists. Add an event causes a new event to be created. `OK` makes the window disappear and generates the event list for the desired event.

Any combination of `shift`, `control`, and `meta` can be selected, but if the `any` modifier button is



**Figure 4-1:** A sample event card deck

selected then the other modifier buttons will become unselected. The mouse actions and keyboard items are all mutually exclusive, so selecting one will cause the previously selected item to be deselected. Events like `#\Return` can be generated by simply typing "Return" in the Specific keypress box (quotes are not needed).

### 4.3. :Start Where

Every interactor dialog box displays two commonly used start-where's for an interactor and allows the user to select an alternative one using the `other` button (Figure 4-2). If `other` is selected, a dialog box will appear which lists all possible `:start-where`'s. Once the desired start-where is selected, Lapidary will incorporate the selected object in the drawing window into the start-where if it is appropriate (which it is in all cases but `t` and `nil`). If the start-where requires a slot (which the `list` start-where's do), Lapidary will request the name of a slot.

If the user wants a type restriction, then pressing the `type restriction` button will cause Lapidary to request a type restriction. A type restriction can be either an atom (e.g., `opal:text`) or a list of items (e.g., `(list opal:text opal:rectangle)`). The type restriction button is a toggle button so if it is already selected, selecting it again will cause the type restriction to be removed. Also, selecting a new start-where will cause the type restriction to be removed.

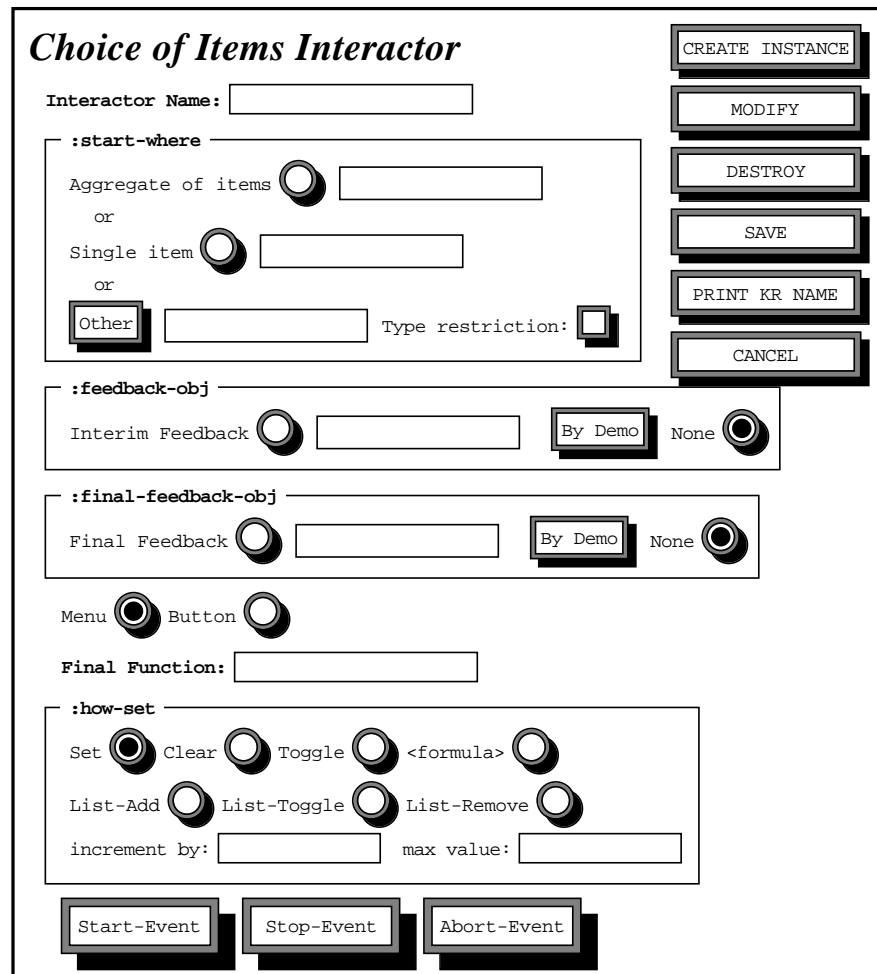
### 4.4. Formulas

Selecting a formula button in any of the interactor dialog boxes causes the interactor to be displayed in the C32 spreadsheet window and the current value of the slot associated with the formula button to be displayed in a C32 formula window. This value can then be edited into a formula. When the `OK` button is pressed in the formula window, C32 disappears and the formula is batched with the other changes that have been made to the interactor since the last `Create Instance` or `Modify` command. The formula is not actually installed until the `Create Instance` or `Modify` buttons are selected. If the user selects `Cancel` in the interactor dialog box, the formula will be discarded. The formula will also be discarded if the user selects `Cancel` in the C32 formula window.

## 4.5. Specific Interactors

### 4.5.1. Choice Interactor

The choice interactor dialog box allows the user to create either a button interactor or menu interactor, depending on whether the menu or button radio button is selected (Figure 4-2). The other slots that can be set using this dialog box are:



**Choice of Items Interactor**

Interactor Name:

**:start-where**

Aggregate of items ☐

or

Single item ☐

or

Other ☐  Type restriction:

**:feedback-obj**

Interim Feedback ☐  ☐ By Demo ☐ None

**:final-feedback-obj**

Final Feedback ☐  ☐ By Demo ☐ None

Menu ☐ Button ☐

Final Function:

**:how-set**

Set ☐ Clear ☐ Toggle ☐ <formula> ☐

List-Add ☐ List-Toggle ☐ List-Remove ☐

increment by:  max value:

Start-Event ☐ Stop-Event ☐ Abort-Event ☐

CREATE INSTANCE  
MODIFY  
DESTROY  
SAVE  
PRINT KR NAME  
CANCEL

**Figure 4-2:** Choice interactor dialog box

- **:start-where.** If the user selects either `aggregadget of items` or `single item` and there is a least one selection in a drawing window (it may be either a primary or secondary selection), then `start-where's` with `:element-of` and `:in-box` are generated with the selected object.
- **:feedback-obj.** Selecting the radio button associated with `interim feedback` will cause the selected object in the Lapidary drawing windows to become the interim feedback for this interactor. If this object is constrained to one of the objects that satisfies the `start-where` or to a component of one of these objects, Lapidary will automatically generalize the constraints so that the object can appear with any of the objects in the `start-where`.

The user can also use the `by-demo` option to demonstrate interim feedback. Lapidary will pop up an OK/Cancel box when an object that satisfies the `start-where` is selected. The user can then use the various Lapidary menus to modify this object so that it looks as it should

when the object's `:interim-selected` slot is set. Once the desired look is achieved, the user selects OK and the changes will be installed so that the object looks like its original self when it is not interim selected, and will look like the by-demo copy when it is interim selected.

Lapidary implements the by-demo operation by comparing the values of the following slots in the original object and the copied object: `:left`, `:top`, `:width`, `:height`, `:visible`, `:draw-function`, `:font`, `:string`, `:line-style`, `:filling-style`, `:x1`, `:x2`, `:y1`, `:y2`.

The last option the user can choose is none in which case nil will be stored in the `:feedback-obj` slot. This will not undo the effects of a by-demo operation since by-demo also places nil in the `:feedback-obj` slot.

- `:final-feedback`. The options for final feedback are identical to those for `:feedback-obj`. The by-demo changes will appear when the object's `:selected` slot is set to `t`.
- `:final-function`. The user can type in the name of a function that should be called when the interactor completes.
- `:how-set`. The user can set the `:how-set` slot by selecting a radio button or entering numbers in the `increment-by` and (optionally) `max value` fields.

#### 4.5.2. Move/Grow Interactor

The move/grow interactor dialog box (Figure 4-3) allows the user to specify a move/grow interactor. The slots that can be set using this dialog box are:

- `:start-where`. If the user selects either `Object to Press Over` or `One of This Aggregate` and there is at least one selection in a drawing window (it may be either a primary or secondary selection), then `start-where`'s with `:in-box` and `:element-of` are generated with the selected object.
- `:line-p`. This slot is set by the `Line` and `Box` buttons. If a formula is selected, the formula should return `t` if the interactor is moving/growing a line, and `nil` if it is moving/growing a box object.
- `:grow-p`. This slot is set by the `Grow` and `Move` buttons. If a formula is selected, the formula should return `t` if the interactor is growing an object, and `nil` if it is moving an object.
- `:min-length`. Specifies a minimum length for lines.
- `:min-width`. Specifies a minimum width for box objects.
- `:min-height`. Specifies a minimum height for box objects.
- `:obj-to-change`. The user can either let the move/grow interactor modify the object that satisfies the `start-where` or use a formula to compute the object to change. The formula can be computed automatically (see below). This slot would be set if the interaction should start over a feedback object such as selection handles, but should actually move the object under the feedback object.
- `:final-function`. The user can type in the name of a function that should be called when the interactor completes.
- `:feedback-obj`. An interim feedback object can be created by creating the desired object and pressing the interim-feedback button. Constraints will be automatically attached to the feedback object that cause it to move or grow appropriately, and that make it visible/invisible at the appropriate times. Alternatively a formula can be entered that selects among feedback objects. Multiple feedback objects can be created by selecting objects one at a time, pressing

the interim feedback button, and then hitting either `create-instance` (the first time, when the interactor is being created) or `modify` (once the interactor has been created). Once the multiple feedback objects have been created, a formula can be entered that selects among the feedback objects.

- `:attach-point`. Controls where the mouse will attach to the object.

The grow and move parameters allow the user to control which slots in the object that is being grown or moved will actually be set. If a formula is entered, it must return a value that can be used by the slot `:slots-to-set` (see the Interactors chapter for more details on this slot).

The user can directly choose an `obj-to-change` in the move/grow and text interactor dialog boxes. Lapidary will automatically construct a formula so that the interactor changes the correct object at run-time. For example, in Figure 4-3, the user wants the move interactor to start over one of the selection handles, but wants the object highlighted by the selection handles moved. The user can specify that the interactor should start over the selection handles by selecting the aggregate containing the selection handles and pressing the `One of This Aggregate` radio button in the move/grow dialog box. The user can specify that the object highlighted by the selection handles at run-time should be the object changed by selecting the example object that the selection handles currently highlight and pressing the `Change this object` button in the move/grow dialog box. Occasionally Lapidary may not be able to determine from the start-where objects which object should be changed at run-time. In this case Lapidary will give the user the choice of entering a formula or of having the example object selected as the `obj-to-change` be the actual object changed at run-time.

Example: To create an interactor that moves a box,

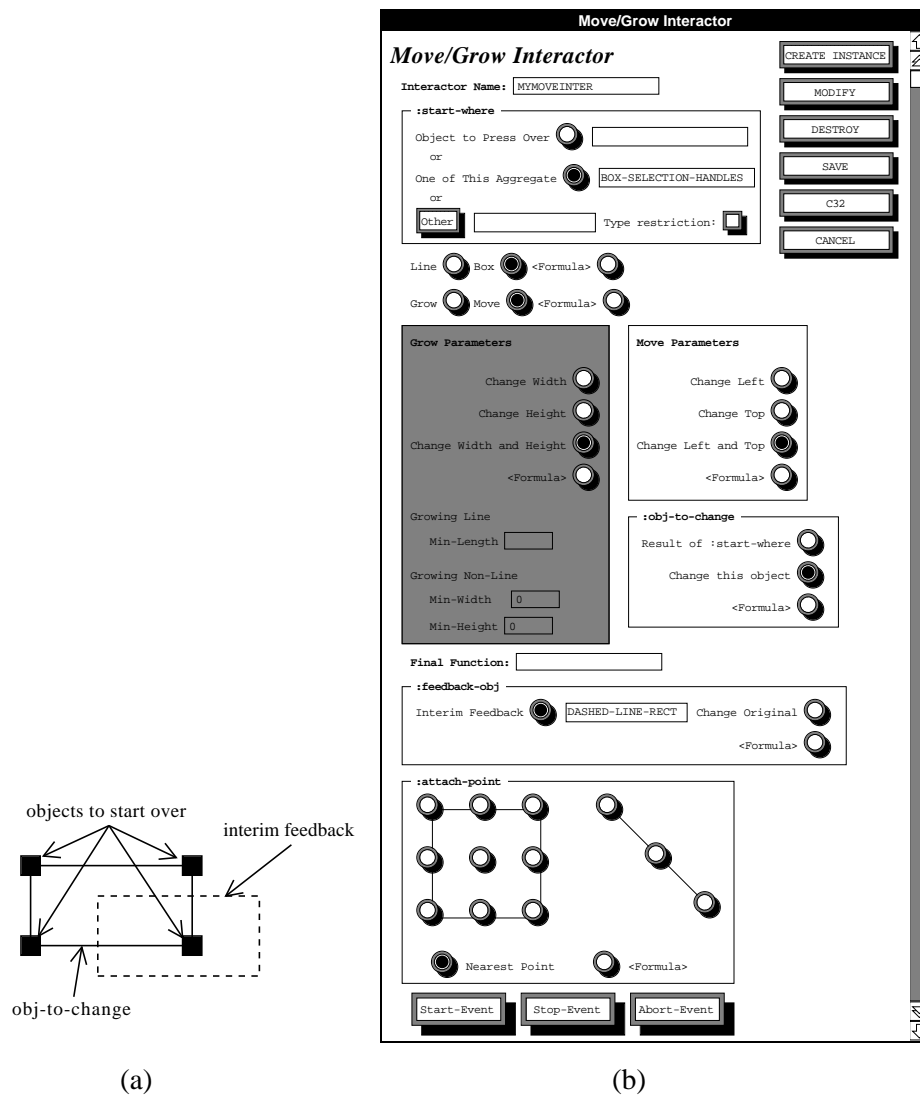
1. Create the box and leave it selected
2. Select interactors from the editor menu and then select move/grow
3. In `:start-where` click on "Object to Press Over". This will cause the selected rectangle's KR name to be displayed.
4. Press the CREATE INSTANCE action button

To test the interactor press test button in the editor menu and drag it around.

### 4.5.3. Two Point Interactor

The two point interactor dialog box (Figure 4-4) allows a user to create a two point interactor. The slots that can be set using this dialog box are:

- `:start-where`. If the user selects `Start Anywhere in Window` then a start-where with `t` is generated. if the user selects `Start in Box` and there is at least one selection in a drawing window (it may be either a primary or secondary selection), then a start-where with `:in-box` is generated with the selected object.
- `:line-p`. This slot is set by the buttons `Create Line` and `Create Non-Line`. If a formula is selected, the formula should return `t` if the interactor is creating a line, and `nil` if it is creating a box object.
- `:min-length`. Specifies a minimum length for lines.
- `:min-width`. Specifies a minimum width for box objects.
- `:min-height`. Specifies a minimum height for box objects.
- `:flip-if-change-side`. Indicates whether a box may flip over when it is being created.
- `:abort-if-too-small`. Indicates whether the operation should be aborted if the object is too



**Figure 4-3:** (a) The various parts for a move behavior; and (b) the dialog box for specifying the move/grow interactor. The user can now directly select the obj-to-change and Lapidary will create a formula that automatically selects the correct object to change at run-time.

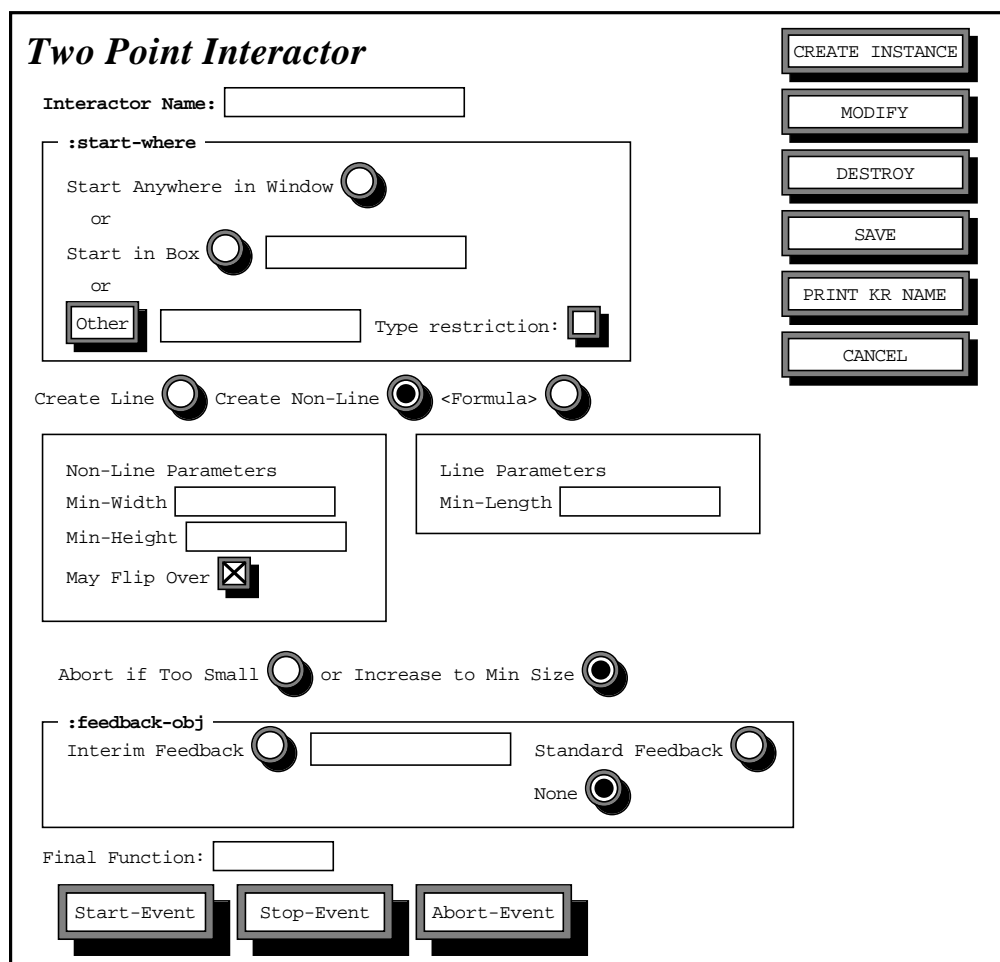
small or whether an object of the minimum size should be created.

- **:feedback-obj.** An interim feedback object can be created by creating the desired object and pressing the interim-feedback button. Constraints will be automatically attached to the feedback object that cause it to sweep out as the mouse cursor is moved, and that make it visible/invisible at the appropriate times. If the standard feedback option is selected, a box or line feedback object is automatically created according to whether a line or box is being created.
- **:final-function.** The user can type in the name of a function that should be called when the interactor completes.

Example: To create a two-point interactor with line feedback

- 1) select the interactors option from `other` in the editor menu and then select two point interactor in the menu that pops up
- 2) click on Start Anywhere in Window
- 3) click on Create Line
- 4) click on Standard Feedback
- 5) click on CREATE INSTANCE

To test this interactor, enter test mode, press down on the left mouse button, and sweep out a line. No line will be created because a final function was not provided.



**Two Point Interactor**

Interactor Name:

**:start-where**

Start Anywhere in Window ☐

or

Start in Box ☐

or

Other ☐  Type restriction:

Create Line ☐ Create Non-Line ☐ <Formula> ☐

**Non-Line Parameters**

Min-Width

Min-Height

May Flip Over ☒

**Line Parameters**

Min-Length

Abort if Too Small ☐ or Increase to Min Size ☐

**:feedback-obj**

Interim Feedback ☐  Standard Feedback ☐

None ☐

Final Function:

**Figure 4-4:** Two point interactor dialog box

#### 4.5.4. Text Interactor

The text interactor dialog box (Figure 4-5) allows the user to create or modify a text interactor and to edit the following slots:

- `:start-where`. If the user selects either object to press over or one of this aggregadget and there is at least one selection in a drawing window (it may be either a



**Text Interactor**

Interactor Name:

**:start-where**

Start Anywhere in Window ☒

or

One of this aggregate ☐

or

Other ☐  Type restriction: ☐

**:object-to-change**

result of start-where ☒

or

<formula> ☐

**:feedback-obj**

Interim Feedback ☐  None ☐

Cursor appears: where pressed ☒ at end of string ☐

Final Function:

Start-Event  Stop-Event  Abort-Event

CREATE INSTANCE  MODIFY  DESTROY  SAVE  PRINT KR NAME  CANCEL

**Figure 4-5:** Text interactor dialog box

primary or secondary selection), then start-where's with `:in-box` and `:element-of` are generated with the selected object.

- `:obj-to-change`. The user can either let the text interactor modify the object that satisfies the start-where or use a formula to compute the object to change. Lapidary can construct a formula for this slot if necessary (see section 4.5.2).
- `:feedback-obj`. An interim feedback object can be created by creating a text object and pressing the interim-feedback button. Constraints will be automatically attached to the feedback object that cause it to appear at the selected text object and that make it visible/invisible at the appropriate times.
- `:cursor-where-press`. This slot is set by the buttons `where pressed` and `at end of string`. If `where pressed` is selected, the text editing cursor will appear under the mouse cursor. If `at end of string` is selected, the text editing cursor will always appear at the end of the string when editing starts.
- `:final-function`. The user can type in the name of a function that should be called when the interactor completes.

#### 4.5.5. Angle Interactor

The angle interactor dialog box (Figure 4-6) allows the user to create and modify an angle interactor. The slots that can be set by this dialog box are:

- `:start-where`. If the user selects `object to press over` and there is a least one selection in a drawing window (it may be either a primary or secondary selection), then a

start-where with `:in-box` is generated. If the user selects start anywhere in window, then a start-where of `t` is generated.

- `:obj-to-change`. The user can either let the angle interactor modify the object that satisfies the start-where or use a formula to compute the object to change.
- `:feedback-obj`. An interim feedback object can be created by creating an object, selecting it, and pushing the interim feedback button. The `:angle` slot of the object will be set as the interactor is operated and the object will be made visible/invisible as appropriate. To make the feedback object or the object that gets the final angle change in response to changes in the `:angle` slot, custom constraints must be created for the position and size slots. See the angle interactor section in the Interactors chapter for sample constraints.
- `:final-function`. The user can type in the name of a function that should be called when the interactor completes.
- `:center-of-rotation`. This is the center of rotation for the interaction. The user can either enter a list of  $(x, y)$ , enter a formula that returns a list of  $(x, y)$  or select one of the standard locations for the center of rotation by selecting the appropriate button.

### Angle Interactor

Interactor Name:

**:start-where**

Object to Press Over ☐

or

Start Anywhere in Window ☐

or

☐ Other  Type restriction:

**:obj-to-change**

Result of :start-where ☐

<Formula> ☐

**:feedback-obj**

Interim Feedback ☐  None ☐

Final Function:

**:center-of-rotation**

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

X  Y

☐ <Formula>

☐ Start-Event ☐ Stop-Event ☐ Abort-Event

CREATE INSTANCE

MODIFY

DESTROY

SAVE

PRINT KR NAME

CANCEL

**Figure 4-6:** Angle interactor dialog box

## 5. Getting Applications to Run

Lapidary-generated files consist of a set of create-instance calls. The objects created are stored in a list and assigned to the variable `*Garnet-Objects-Just-Created*`. The top of a Lapidary-generated file contains code to load the `lapidary-functions.lisp` file, which provides functionality to support the created objects.

# Index

- Add gadget 545
- Aggregadget, make 551
- Aggregadget, selection 540
- Aggrelist 539
- Arrange 551
  
- Bitmap 539
- Box constraint gadget 556
- Box constraints 551
- Box-constraint-do-go 560
- Bring to front 551
- Build 552
  
- C32 551
- Choice interactor 565
- Circle 539
- Clear workspace 552
- Constraint-gadget 556
- Constraints 551, 553
- Create object 542
  
- Delete object 546
- Delete window 546
- Draw function 546
  
- Edit 546
- Events 563
- Exit 546
  
- Filling style 546
- Formulas 564
  
- Grow 541
  
- Install-links 561
- Interactors 551, 563
  
- Line 539
- Line constraint gadget 559
- Line constraints 551
- Line style 546
- Line-constraint-do-go 560
- Link slots 561
- List properties 547
  
- Make copy 546
- Make instance 546
- Mouse 540
- Move 541
- Move/grow interactor 566
- Multi-text 539
  
- Name object 546
  
- Other, menu selection 551
  
- Parameters 547
- Primary select covered object 541
- Primary selection 540
- Primary Selection, add to 541
- Primary selection, deselect 541
- Properties 546
  
- Quit 546
  
- Rectangle 539
  
- Resize 541
- Roundtangle 539
- Run Lapidary 539
  
- Save 545
- Secondary select covered object 541
- Secondary selection 540
- Secondary selection, add to 541
- Secondary selection, deselect 541
- Selecting objects 540
- Send to back 551
- Show-box-constraint-menu 560
- Show-line-constraint-menu 560
- Start Lapidary 539
- Stop 546
  
- Test 552
- Text 539
- Text interactor 569
- Text properties 547
- Text, edit 542
- Two point interactor 567
  
- Ungroup 551
  
- Window 539

# Table of Contents

<b>1. Getting Started</b>	<b>539</b>
1.1. Object Creation	539
1.2. Selecting Objects	540
1.3. Mouse-Based Commands	540
1.4. Selection Techniques	542
<b>2. Editor Menu Commands</b>	<b>545</b>
2.1. File	545
2.2. Edit	546
2.3. Properties	546
2.4. Arrange	551
2.5. Constraints	551
2.6. Other	551
2.7. Test and Build Radio Buttons	552
<b>3. Creating Constraints</b>	<b>553</b>
3.1. Box Constraints	553
3.2. Line Constraints	555
3.3. Custom Constraints	556
3.4. The Constraint Gadget	556
3.4.1. Box Constraint Menu	556
3.4.2. Line Constraint Menu	559
3.4.3. Programming Interface	559
3.4.3.1. Slots of the Constraint Gadget	559
3.4.3.2. Exported Functions	560
3.4.3.3. Programming with Links	561
3.4.4. Custom Constraints	561
3.4.5. Feedback	562
<b>4. Interactors</b>	<b>563</b>
4.1. Action Buttons	563
4.2. Events	563
4.3. :Start Where	564
4.4. Formulas	564
4.5. Specific Interactors	565
4.5.1. Choice Interactor	565
4.5.2. Move/Grow Interactor	566
4.5.3. Two Point Interactor	567
4.5.4. Text Interactor	569
4.5.5. Angle Interactor	570
<b>5. Getting Applications to Run</b>	<b>573</b>
<b>Index</b>	<b>574</b>