A Drawing Application for Curl

by

Arthur E. Housinger

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

Author .....
Department of Electrical Engineering and Computer Science
May 26, 1998

Certified by .....
Stephen A. Ward
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Drawing Application for Curl

by

Arthur E. Housinger

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 1998, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

Curl is a new Web devolopement language. As a new language, there is a learning curve necessary for users to overcome before being able to create interesting web pages.

This thesis introduces Canvas, an application for creating web pages written in Curl. The strength of Canvas is two fold. First, it is a visual tool that keeps the Curl code as far away from the user as possible. Second, Canvas is written to facilitate additions to it. As Curl evolves and more features are added to it, the Canvas may too be updated.

Thesis Supervisor: Stephen A. Ward
Title: Professor

Acknowledgments

I would like to thank my parents for their love and support throughout my life. They have made me what I am today. A big thank you also goes out to all my friends who ignored me in the final weeks before this thesis was due. I would like to thank Steve Ward for his patience and positive feedback every time we talked. Finally, I would like to thank Rena Yang for her input both on the design and on the paper, for her encouragement the past few months, and for taking care of the "little things" while I worked.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Curl is an object–oriented language for creating web documents. Curl is intended to be a gentle–slope system, which is defined as a system "accessible to content creators at all skill levels ranging from authors new to the web to experienced programmers".[8] In accomplishing this task, Curl encompasses a large body of functionality currently found in HTML, Tcl/Tk, Java^{TM}, and TeX^{TM}.[1]

However, as with any new language, Curl requires its users to learn a new syntax and become familiar with built–in classes. New users find themselves reading specifications, source code, and tutorials. For authors new to the web, this may be daunting. In addition, since Curl is still in its Alpha stages, the specifications for classes are changing. This can lead to difficulties when programming in Curl.

To help programmers avoid the obstacles of learning a new computer language, many applications have been created that allow the programmer to directly manipulate the visual layout, rather than the code of the language.[7, 11, 13, 14, 15] In addition, these applications may remind the user of different objects' specifications and often fill in attributes of the objects with defaults for the user.

[1] Java is a trademark of Sun Microsystems, Inc. in the United States and other countries. TEXis a trademark of the American Mathematical Society.

1.2 Project Overview

This thesis presents the framework for an application, entitled Canvas, that helps a user create web pages in Curl. The purpose of this application is to reduce the learning curve for Curl, as well as to facilitate quick creation of Curl pages. Since Curl is still evolving, and since Curl may be extended by its users, Canvas is written to allow programmers to easily extend its functionality.

In addition, a drawing mechanism is added to Canvas. This mechanism serves three purposes. First, it tests the correctness of Canvas. Second, it tests the usability of Canvas. Third, it gives Curl users a way to draw without having to manipulate pixel values, which is currently the only way to draw in Curl.

The rest of the thesis is laid out as follows. Chapter 2 explains briefly the drawing and layout mechanisms for Curl used in this thesis. Chapter 3 talks about other applications that influenced the design of Canvas. Chapter 4 discusses the user interface for Canvas, and includes an example of its use. Chapter 5 discusses the internal classes and procedures that are used to create the Canvas application, as well as the philosophy behind them. Chapter 6 discusses the two file formats used to save work done in Canvas. Chapter 7 discusses future work to be done for both Canvas and Curl. Finally, Chapter 8 reviews the successes and failures of the Canvas with respect to its goals.

Chapter 2

A Brief Curl Introduction

There are several mechanisms that Curl supplies for laying out a web page. Currently, however, Canvas only takes advantage of two: Frame objects and the draw method.

An object of type *Frame* contains objects that inherit from the class *Graphic*. Frames were used by Canvas because they allow the positions of contained objects to be specified. For example, in Figure 2-1 the Button (which inherits from Graphic) is first placed in the Frame, and then moved to a location in the middle of the Frame. This flexibility was needed for placing the different objects created in Canvas.

The second mechanism, the draw method of a Graphic object, is automatically called by the Curl browser. This method is passed a GraphicContext, which has many methods to facilitate drawing, including the following: set–color, set–font, draw–line, draw–rect, and draw–text. By calling these methods, an object can draw in its allocated space on the screen.[4]

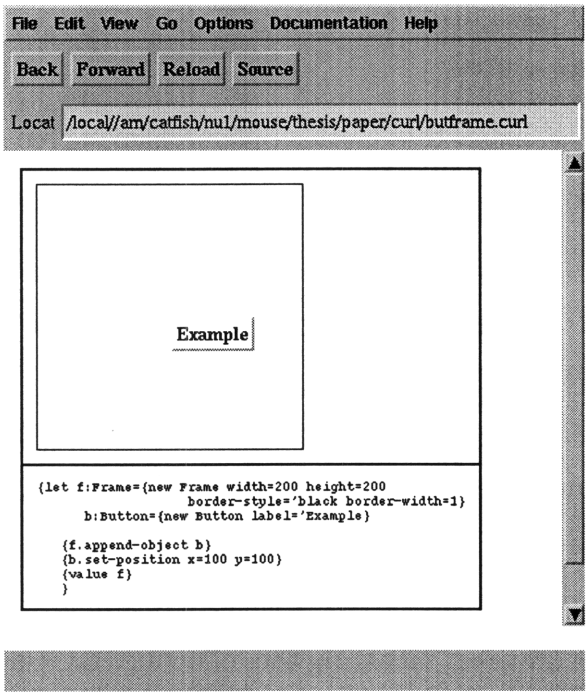


Figure 2-1: Placing a Button object inside of a Frame object

Chapter 3

Related Work

Many applications were researched for ideas of what to include, what not to include, what complements a gentle slope language, and what is difficult to learn. Of the applications researched, two categories arise. First, there are applications meant for creating a presentation of sorts. Second, there are applications that are meant specifically for drawing. Although presentation applications often support drawing, and drawing applications often support presentations, the main focus of the applications are used to categorize them.

3.1 Presentation Applications

Three presentation applications were researched: HoTMetaL PRO^{TM}, PowerPoint®, and FrameMaker®.[1][9, 10, 11] These three applications are used most often to respectively create web pages, create business presentations, and create text documents.

HoTMetaL PRO gave insight into what features must be contained in an application used to create web pages. HoTMetaL PRO offers its users several text formatting options, a table editor, and a forms and widget editor. The widgets can be edited, via a dialog box, to perform actions or contain code. The dialog box offers a gateway to code, without completely forsaking the visual power of a layout application. Un-

[1]HoTMetaL PRO is a trademark of SoftQuad Inc. PowerPoint is a registered trademark of Microsoft Corporation. FrameMaker and Adobe are trademarks of Adobe Systems Incorporated.

fortunately, the user interface is difficult to utilize efficiently; its numerous buttons slow down the application considerably, and they slow down the user in trying to find a desired button.

PowerPoint offers a more powerful presentation application. The numerous tools offered are organized into a hierarchy of menus. In addition, it offers a set of page templates. Items, such as text or a polygon, can be moved, reshaped, rotated, and aligned. However, PowerPoint has two main drawbacks as a model. First, it only allows a user to create objects visually; creating a line that is exactly four inches long is impossible. Secondly, it forces a style of creation for the user. For example, after drawing a line, the user is forced to be in *select mode*, where they can move the line, reshape the line, or change the attributes of the line. However, in this mode, the user cannot draw another line; they must reselect the *line drawing mode*. While this may be useful when creating a document with only a few lines, it is an annoyance when creating a page made of many lines.

FrameMaker software offers a wide variety of tools, including drawing capabilities and equation editing. Each tool offers its own separate pop–up toolbar. This is convenient because it keeps toolbars separate, and it allows new features to be easily added by Adobe®. FrameMaker software, however, has the same problems that we mentioned PowerPoint having. (Although rulers make "exact" drawing more feasible.) Many small stylistic attributes of the FrameMaker and PowerPoint (such as being forced into select mode) are similar, and were used extensively as guidelines when coding this thesis.

3.2 Drawing Applications

There were two classes of drawing applications: pixel based and object based. Pixel based applications allow manipulation of each pixel separately. Although you can draw objects, once drawn, there is no relation between the pixels. Object based drawings allow the user to manipulate the different objects, such as lines, rectangles, and circles.

3.2.1 Pixel Based Applications

A pixel–based approach to drawing is not acceptable for Canvas, since Canvas is meant for creating Curl code. A pixel–based drawing application for Curl would only serve in creating bitmaps added to a Curl page, and this goal is better served by drawing the pictures in other applications and exporting to a bitmap for inclusion on a Curl web page. Therefore, many lessons offered by pixel–based applications[1, 6] were unusable. However, both user interface information and some features of the applications were considered.

For example, SuperPaint® for the Macintosh® has four ways to resize a rectangular region.[2] First, you can "scale" the region, which changes the length of all four sides of the region. You can use "slant" to keep one corner fixed, while changing the positions of the other three corners. "Distort" keeps two corners intact, while the other two move. Finally, "stretch" keeps three corners fixed while the last corner moves. Canvas was designed to allow any reshaping of the objects within it. Although objects are not pixel based, the aforementioned reshaping methods are all possible within Canvas.

3.2.2 Object Based Applications

A number of object based drawing applications were viewed during the design phase of this thesis, and attributes were taken from each. The applications researched included Canvas (the original drawing application in Curl), xfig, Canvas^{TM} (an application for the Macintosh)[3], Sketchpad, and AutoCAD®.[4][2, 5, 12, 16]

Curl's original Canvas application offers some components for reuse. However, much of its organization did not allow for the versatility desired. Therefore, most of the original work was discarded.

[2]SuperPaint is a registered trademark of Aldus Corporation. Macintosh is a registered trademark of Apple Computer Inc.

[3]For clarity, the phrase "original Canvas" is used throughout to refer to the first drawing application in Curl, and the phrase "Deneba's Canvas" is used to refer to the application for the Macintosh. All other uses of the phrase "Canvas" refer to the application presented in this thesis.

[4]Canvas is a trademark of Deneba Systems Incorporated. AutoCAD is a registered trademark of Autodesk, Inc.

The application xfig was most useful because its source is available.[16] The mechanism in Canvas for handling mouse events was inspired by xfig. In addition, the undo mechanism in xfig was mimicked. Finally, much of the layout of xfig is simple and communicates easily to the user, and was therefore partially adapted. However, xfig is very limited, both in what it can do and also in features that it is capable of adding, and care was taken not to copy xfig in any way that might limit the functionality of Canvas.

Deneba's Canvas application for the Macintosh was interesting in that it has some features that none of the other applications have. For example, it allows the user to "snap to grid" in the x direction only, y direction only, or both. (Most application require both or none.) It also allows users to define their own line type and arrow type. This thesis attempted to mimic the versatility of Deneba's Canvas.

Sketchpad is a drawing application created at Lincoln Laboratories in 1963. Despite being created decades ago, it has one feature that is not found in current applications: the ability for objects to mimic each other. For example, you can draw a bolt and place copies of that bolt in hundreds of locations in your drawing. Then, if the specifications for the bolt changed, you could change just one of the bolts, and the rest would change automatically. This feature is supported by Canvas.

AutoCAD contains, by far, the most drawing options. For example, a user can draw an arc eleven different ways in AutoCAD. However, this versatility can be daunting and confusing to a new user, an undesirable feature for a gentle slope language. At the same time, the power that it offers was desirable. (Allowing a dialog for each modes in Canvas was inspired by the versatility of AutoCAD.) In addition, since AutoCAD is so large, it attempts to be as clear as possible. For example, when drawing a line, a button press is requested for both points. Finally, AutoCAD attempts to give the user several "shortcuts" to different features; a user can draw a line by typing "line" at the text prompt, by going to "Line" in the menu bar, or by going to "Line" on the dynamic menu bar. Canvas attempts to capture both the clarity and the versatility of AutoCAD.

Chapter 4

Graphical User Interface of Canvas

To use Canvas, a user creates a Curl web page that contains an object of type CanvasApplication. This object, then, will then be laid out as shown in Figure 4-1. The application, from the user's standpoint, may be divided into six main parts: the drawing sheet, the mode toolbar, the PopupDialog button, the property toolbar, the menu, and the message windows.

In the first section, we define the parts and explain how each is used. In the second section, we demonstrate their use.

4.1 The Parts

The drawing sheet is the large square in the center of the Canvas application. This is where all drawn figures and all created objects appear. In addition, the rulers on the sides of the sheet track the current mouse position. Users may switch between different drawing sheets using the menu choice *Windows*.

The mode toolbar is located to the left of the drawing sheet. This toolbar may dynamically change to assist in different modes (such as table creation, drawing, and text layout). Currently, however, the only available mode toolbar is for drawing. This toolbar is made up of three sections: Draw, Edit, and Action. Only one button found in the Draw and Edit sections may be selected at a time, and the selected button defines the interpretation of mouse events within the drawing sheet. Buttons in the

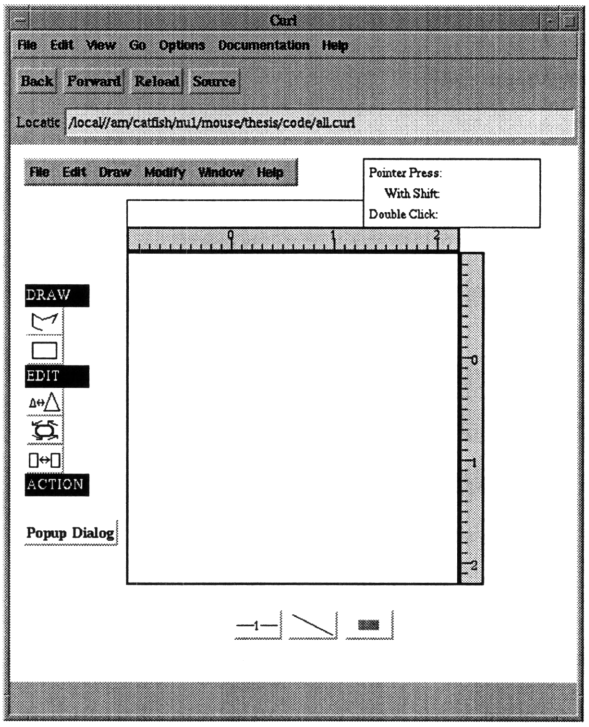


Figure 4-1: The initial Canvas.

Action section perform functions, such as flipping horizontally or vertically. Once finished with their function, control returns to the currently selected Draw or Edit button. (Currently, no action buttons are implemented.)

Immediately below the mode toolbar is a button labeled "Popup Dialog". When pressed, this button causes a dialog box to pop up for the currently selected mode. The dialog box gives the user a text interface to the current mode.

The property toolbar is located below the drawing sheet. This toolbar is used to change properties for Canvas such as line width, line color, and fill color. Each property button, when clicked, invokes a dialog box for the user to change the property. As with the mode toolbar, the property toolbar may display a different selection of buttons during a Canvas session.

The menu is located in the upper–left corner of the Canvas application. In addition to providing options not available elsewhere in Canvas, it provides another means for switching modes and changing properties. For example, by choosing the menu option

"Polyline" under the menu heading "Draw", the current mode will change to the polyline mode and the mode toolbar will reflect this change.

The message windows are located above the drawing sheet. The message window on the right is used to convey to the user what effect the mouse has in the current mode. The message window on the left is used to communicate with the user in a unobtrusive way.

4.2 An Example

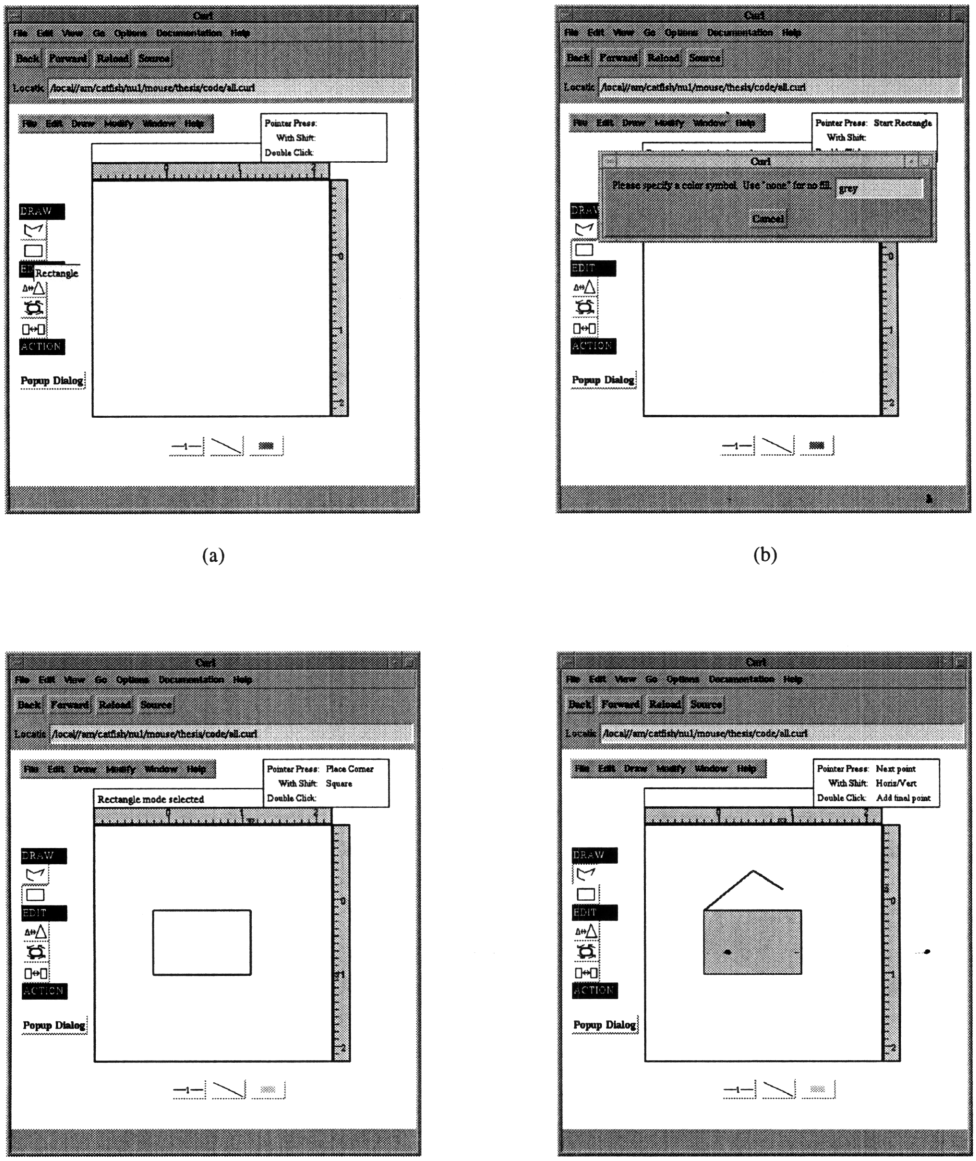
Before discussing the internal components of Canvas, an example of its use will be presented. This example does not extend the functionality of the current Canvas, nor does it discuss how programmers may do this. In addition, because we focus on graphical user interface in this chapter, no mention is made of what happens internally during the example. The example chosen is drawing a simple house, rotating it, and exporting it.

To begin drawing, we select the mode for drawing a rectangle, as shown in Figure 4-2a. Next, in figure Figure 4-2b, we select the fill color and change its value to "grey". Then, in Figure 4-2c, we draw the body of the house. Finally, in Figure 4-2d, we finish the house by changing to the polyline mode and drawing a simple roof over the house.

At this point, all drawing is done, and we begin editing. First, we choose rotate. As shown in Figure 4-3a, Canvas has automatically selected our last drawn object as the object to rotate. However, we wish to rotate both the house body and the house roof; therefore, we draw a select box around both objects, as shown in Figure 4-3b. At this point we can rotate. However, when editing multiple objects at once, each object is edited separately. Therefore, when rotating the two parts of the house, the body and the roof rotate about their individual centers. Each object is rotated by the same degree, but, as shown in Figure 4-3c, the final rotated product is not what we desire. To correct this, we switch to move mode, in order to move the roof. Since two objects were just selected, Canvas assumes that we want to move both objects.

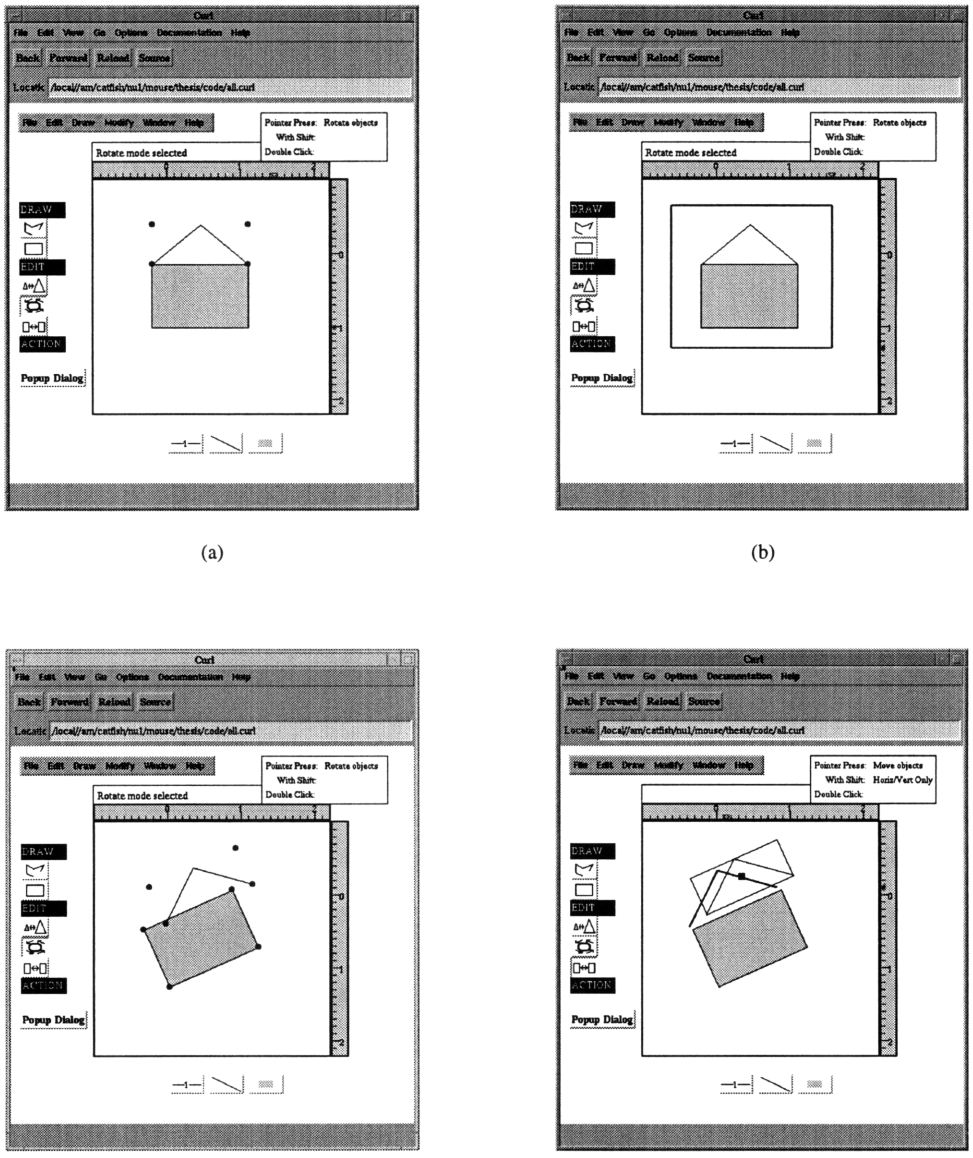
We select just the roof (by first clicking away from all objects and then clicking on the roof) and move the roof to its proper location, as shown in Figure 4-3d.

Our house is now rotated. We now decide to make the house larger. Therefore, we change to resize mode, select both objects again, and begin resizing. Notice in Figure 4-4a that although the house is rotated, it may be expanded "to the right" as though it has not been rotated. This feature may be turned on or off, but for this example we will use it. Finally, we export the house to a file (we are prompted for the filename) to be included in a Curl web page. (See Figure 4-4b and Figure 4-5.)



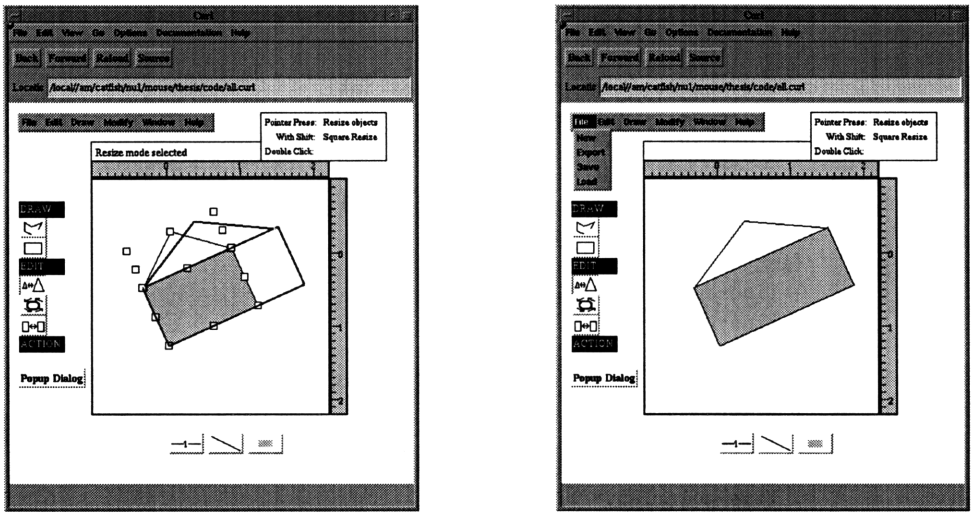
In part a, the user is about to choose the mode to draw a rectangle. In part b, the user is in the process of changing the current fill color. In part c, the user is in the process of drawing the rectangle. In part d, the user is in the process of drawing a polyline.

Figure 4-2: Drawing in Canvas.



In part a, the user has selected the rotate mode. The polyline is ready for rotation. In part b, the user is selecting both the polyline and the rectangle using a select box. In part c, the user has finished rotating both objects. In part d, the user is moving the polyline.

Figure 4-3: Rotating and Moving in Canvas



(a) (b)

In part a, the user is resizing both the rectangle and the polyline. In part b, the object is finished, and the user is about to export the picture to a file.

Figure 4-4: Resizing and Exporting in Canvas

```
| File created by Curl's Canvas Application


| And now everything drawn in the Canvas

{define-class class-number-1 {SimpleGraphic}
   width:int
   height:int

  {define {init}
    {set self.width 322}
    {set self.height 322}
    {super.init}
    } | method: init

  {define public {layout}:{return int int}
    {return self.width self.height}
    } | method: layout

  {define public {draw gc:GraphicContext}:void
    | CanvasLineObject stuff
    {let
      line-color:any='black
      line-width:int=1
      {gc.draw-line 68 148 137 57 line-color line-width}
      {gc.draw-line 137 57 248 67 line-color line-width}
    } | let

    {let
      line-color:any='black
      line-width:int=1
      | rotated rectangle
      | Start with the filling
      {gc.fill-polygon 'grey 67 148 251 65 287 144 103 228}
      | and now the border, which may overwrite filling
      {gc.draw-line 67 148 251 65 line-color line-width}
      {gc.draw-line 251 65 287 144 line-color line-width}
      {gc.draw-line 287 144 103 228 line-color line-width}
      {gc.draw-line 103 228 67 148 line-color line-width}
    } | let


    | Finally, a call to super.draw
    {super.draw gc}
    } | method: draw
  } | class class-number-1
```

The resulting exported file for the example in Section 4.2.

Figure 4-5: Exported Code

Chapter 5

Canvas Internals

Canvas's internal design is divided into the following parts: the objects, the modes, the selection boxes, the five externally visual parts, and the application as a whole. In this chapter, we will describe the purpose of each part. In addition, key methods for achieving these purposes will be explained. Finally, this chapter explains some of the advantages and disadvantages of the current organization. The actual code may be viewed in conjunction, and may be found in Appendix A.

5.1 Canvas Objects

All objects found in the drawing sheet must inherit from class NewCanvasObject. These objects contain all necessary information to be drawn, selected, and exported by Canvas. Objects of type NewCanvasObject may be queried for attributes by other components of Canvas. The objects may also be modified. Both queries and modifications are accomplished using the methods four–corners, encloses–object, partially–encloses–object, add–to–canvas, remove–from–canvas, reshape, rotate, distance–to–object, copy, change–object–property, save–to–file, read–from–file, and export–draw. Finally, public member variable marker–type may be used by selection boxes and markers. (See Section 5.3 for more information.)

The method four–corners is used to identify the four corners of the object. If the object is unrotated, this would be the corners of the Graphic object from which

the object inherits. However, if rotated, this method allows the object to define its rotated bounding box.

The methods encloses–object and partially–encloses–object are used for selection purposes. They are passed two points, representing the corners of an unrotated rectangle on the screen. The object returns whether the rectangle fully encloses the object (in the case of encloses–object) or whether the rectangle encloses any part of the drawn object (in the case of partially–encloses–object).

The methods add–to–canvas and remove–from–canvas are used to place and remove an object from a drawing sheet. An object is given control of adding and removing itself to the drawing sheet, so that any special actions that it may require can occur. However, if the object must be added through other means, the method added–to–canvas should be called.

The reshape method is used to modify an existing object. It is passed a function that takes a point and returns a new, modified point. Depending on other arguments, an object may either apply this function to a single point or to all points that define it. The generic nature of reshape allows for objects to be able to "stretch", "distort", and "slant" (as defined in Chapter 3) without understanding the details of these operations. In addition, the generality of reshape allows new ways of reshaping to be added to Canvas in the future. The reshape method should be used for all reshaping except for rotation. Rotation should be done solely by calling the rotate method.

The rotate method takes an angle for the object to rotate. This method is separate from reshape so objects may retain an orientation to the Canvas application. This lets a user rotate a rectangle and then move the "left side" of the rectangle, as demonstrated previously in Figure 4-4.

The method distance–to–object is called to determine how far away a mouse click was from the visual representation of the object. The object takes into account the fact that it does not draw in all of its allocated space.

The copy method is used to make a copy of a NewCanvasObject. This is currently used extensively by undo and redo. See Section 5.4 for more information on this use.

The method change–object–property allows for visual attributes of the object

to be changed. Visual attributes include line color, line width, and fill color. If an attempt is made to change a property not previously added by the object, the call to change–object–property is ignored. This is because a property not added by the object cannot be utilized by the object. Therefore, adding the property would needlessly take up memory in the object, and it might mislead a programmer to think the property is used.

The next two methods, save–to–file and read–from–file, are used during the saving and loading of a Canvas file. The object stores all internal state information when saving. When loading, the object will reinitialize all variables. This design allows the object to modify how it stores its state, without having to change other code.

The final method, export–draw, is only found in objects of type NewCanvasDrawingObject. This method assume that it is invoked while a draw method is being written to the output file. In other words, this method is used to export code that creates an image of the object using methods available in the GraphicContext.

5.2 Canvas Modes

All modes for Canvas must inherit from class NewCanvasEventHandler. The mode will be passed events from the drawing sheet. The mode then interprets the events and may draw on the drawing sheet or add objects to the drawing sheet. The following paragraphs describe the interface to a NewCanvasEventHandler.

The methods pointer–motion, pointer–release, pointer–press, and pointer–dbl–clk are passed two items: the original PointerEvent for the mouse event and a point describing the "snap coordinates" of the event.[1] Each mode may then decide if it is more appropriate to use the snap coordinate or the original position.

The method key–press receives all keyboard input with the exception of the ESC key. An ESC key press causes a call to esc–key–press, which by default calls method

[1]For those unfamiliar with snap, snap rounds the coordinates of mouse events to the nearest point that is a multiple of the snap size. Snap size may usually be set by the user. Snap coordinates refers to the position of the new point, after rounding the coordinates.

cancel.

When method cancel is invoked, a mode clears any visual changes that it has made to the drawing sheet. In addition, it may release any memory that it would unnecessarily be keeping. Similarly, the method named "selected" is invoked to notify a mode when it is selected, allowing the object to reinitialize and, if necessary, reallocate some memory for internal variables.

Finally, popup–dialog–box creates a dialog box for the mode. This box allows text input for the mode, facilitating precise creation or precise manipulation of objects.

5.3 Selection Boxes and Markers

5.3.1 Boxes

Selection boxes are the boxes that appear around a selected object. Selection boxes are of type CanvasBox, a subclass of NewCanvasObject; however, they are different from NewCanvasObjects for several reasons. First, items of this type are always visually on top of the other items in a drawing sheet. Second, these items are not allowed to add themselves to the drawing sheet. (An error will signal if add–to–canvas is called.) Third, selection boxes always visually surround a NewCanvasObject. Finally, and most importantly, many selection boxes always appear with "markers" of type NewCanvasSelectMarker. Markers are drag–able objects that are used to modify objects within select boxes.

Selection boxes have one important public method, change–object. This method, when invoked, changes which object the CanvasBox surrounds and changes. In this way, a single selection box may be recycled, and hence avoid the costs of reallocation.

5.3.2 Markers

Markers, or objects of type NewCanvasSelectMarker, are drag–able objects. They are the only objects placed in the drawing sheet that respond directly to mouse events. For this reason, very few of its methods need to be called externally. Instead, a marker

reacts to drag events directly, and calls its own methods to react. Specifically, when being dragged, a marker calls its update–dragging method, and when released, the method done–dragging is called. In both methods, the marker then performs three actions: it updates its internal state, it locates objects to modify, and it changes these objects. These three actions are the focus of the following paragraphs.

How a marker updates its internal state is specific to the marker. However, the general mechanism can be shown by using a specific marker as an example, and RotateBoxMarker is chosen. When dragged, a rotation marker first finds its pixel offset from its original location. From this information, it then calculates the angle defined by the three points

- the original position of the marker
- the center of the marker's associated NewCanvasObject
- the new position of the marker

with the object's center being the vertex of the angle.

After changing internal state, a marker finds all selected objects that respond to its intended changes. To determine all objects that respond to a particular action, the marker looks at the member variable *marker–type* for each NewCanvasObject selected. Each NewCanvasObject uses this variable to declare what actions may be performed on it. This mechanism has two strengths. First, it allows objects, such as text, to prevent impossible actions from being attempted, such as rotation. Second, it allows for new types of markers (and new types of object modifications) to be added to Canvas without all objects having to know about the new marker.

Finally, once all changeable objects are found, a marker will modify the objects. This is done indirectly through either an object's reshape method or its rotate method. The word indirectly is used because the marker will actually call the drawing sheet's method update–object–for–undo, which is discussed in the next section.

5.4 The Visual Parts

The drawing sheet is an object of type NewCanvasSheet. The drawing sheet and the Canvas application share a role as a central point for all components to report. The drawing sheet receives all queries and actions that effect objects within the drawing sheet, while the Canvas application receives all queries and actions that effect objects outside the drawing sheet. In this section, we will first discuss the methods available in NewCanvasSheet. We then discuss the relation between the other visual parts, the drawing sheet, and the Canvas application.

5.4.1 The Drawing Sheet

NewCanvasSheet has many methods for assisting the other components of Canvas. These include change–MEH, add–select–box, remove–select–box, add–canvas–object, remove–canvas–object, hide–layers, show–all–layers, closest–object, exchange–canvas–objects, commit–exchange, update–object–for–undo, undo, redo, get–object–CanvasBox, popup–MEH–dialog, add–canvas–property, change–canvas–property, and get–canvas–property.

The first method, change–MEH, changes the current NewCanvasEventHandler, which is the event handler for the current mode. As you will recall from Section 5.2, the NewCanvasEventHandler is passed all keyboard and mouse events that are sent to the drawing sheet. Before changing modes, this method invokes the current mode's cancel method. In this way, the different modes do not have to communicate with each other.

The two methods add–select–box and remove–select–box are used directly to add objects of type CanvasBox. For example, a NewCanvasEventHandler that just created a new selection box would then call add–select–box to add this box to the drawing sheet. This way, the NewCanvasSheet has control and may add and remove other selection boxes, without the knowledge of the newest selection box. The methods add–canvas–object and remove–canvas–object, however, should be called only by the NewCanvasObject being added or removed. This allows the objects to do any special

initializing.

The method hide–layers can be used to show only a sub–set of the objects added to the drawing sheet. This method will also set internal state of the drawing sheet to reflect that layers are hidden. This setting then allows the drawing sheet to act appropriately when the current depth is changed or when new objects are added. The effects of hide–layers may be undone with a call to show–all–layers.

The method closest–object takes a point and returns the object closest to that point. This method gives quick access to the currently added objects and is used to aid in selection.

The methods exchange–canvas–objects and commit–exchange are used to allow actions to be reversed. The former takes two sets of objects, those to be removed and those to be added, either of which may be empty sets. In addition, you may specify that the exchange is part of a larger set of exchanges, to be undone with one call. In this case, a call to commit–exchange may be used to signal the end of the exchange.

One may also use update–object–for–undo. This method takes an object and a method name. It will make a copy of the object, invoke the passed method of the copy, and then exchange the new object for the original. The original object is then stored, and may be switched with the new object to reverse the effects of the update. This mechanism for undo does use extra memory. However, the alternative protocol would not copy the object, but require each object to be able to undo the last modification to it. The extra use of memory is a much smaller cost compared to the complexity the alternative would add.

The method undo may be invoked consecutively multiple times. Hence, the user may undo several actions. Currently, the undo buffer size is set at ten. A call to redo has effect only if no objects have been added since the last undo.

The method get–object–CanvasBox allows for an easy way to see if a particular NewCanvasObject is currently selected. If it is, the CanvasBox that surrounds the object will be returned. Although this is slower than storing a pointer to the CanvasBox in the NewCanvasObject, there is no reason for the NewCanvasObject to know when it is selected, or even to know about selection boxes.

The next method, popup–MEH–dialog, allows the PopupDialog button in the mode toolbar to have minimal knowledge. Without this method, the button would have to have knowledge of the current selected NewCanvasEventHandler.

Finally, the three methods add–canvas–property, change–canvas–property, and get–canvas–property allow manipulation of the properties known by the drawing sheet. These properties are the visual properties of objects, such as line color, line width, and fill color. The method add–canvas–property allows for new properties to be added by programmers who extend Canvas. One method could have been used for both add–canvas–property and change–canvas–property, but the purpose of invocation is unclear in such a design.

5.4.2 The Other Visual Components

As mentioned at the beginning of this section, each visual component of Canvas (other than the drawing sheet) use the drawing sheet and the application itself as connections to the other components of the application. Other than these connections, the components are as separated as possible.

The message windows are the most separate from the other components of Canvas. They communicate no information to any other part of Canvas, and they receive information from the other parts only through methods supplied in the CanvasApplication.

The menu is connected both to the CanvasApplication and to the NewCanvasSheet. Using methods in the CanvasApplication, programmers may add items to the menu or remove items from the menu. Invocation of the menu items cause the current NewCanvasSheet to be passed to the associated procedure for the menu item. This allows menu items to change the state of the drawing sheet, such as changing properties, copying objects, and removing objects.

New buttons may be added to either toolbar using methods available in the CanvasApplication. (See Section 5.5.) In addition, either toolbar may be switched with another toolbar, allowing, for example, a mode bar for Tables and a mode bar for drawing. Once toolbars are in place, they communicate with the drawing sheet rather than the application. Mode buttons, when selected, either call the NewCanvasSheet's

change–MEH method or perform their action on currently selected items in the drawing sheet. Property buttons, when selected, call change–canvas–property with the new value entered by the user.

5.5 The Application

The Canvas application is an object of type CanvasApplication. As already mentioned, the application itself is used as a central location for components to communicate with other components. In this section, we outline the different methods available to access the different parts of the Canvas application. We will discuss the methods CanvasApplication add–canvas–sheet, set–current–sheet, remove–canvas–sheet, make–new–sheet, add–menu–heading, add–menu–item, remove–menu–item, add–toolbar, change–toolbar, add–toolbar–button, set–error–msg, set–message, set–event–descriptions, add–semaphore, check–semaphore, lock–semaphore, release–semaphore, set–canvas–property–button, add–application–property, get–application–property, and change–application–property.

The first four methods, add–canvas–sheet, set–current–sheet, remove–canvas–sheet, and make–new–sheet, allow the user to open multiple files simultaneously and to use multiple drawing sheets. The next three, add–menu–heading, add–menu–item, and remove–menu–item, allow a programmer to dynamically change the mode bar.

The next three methods, add–toolbar, change–toolbar, and add–toolbar–button, allow a programmer to change both the mode toolbar and the properties toolbar. The first method will add a toolbar to the list of known toolbars. The second allows the programmer to change which toolbar is currently visible. This may be used if there are two modes toolbars, such as one for tables and one for pictures. The last method is used to add buttons to any of the toolbars.

The methods set–error–msg, set–message, and set–event–descriptions are used to update the message boxes. Each method simply passes its arguments onto the message boxes. Although this adds a layer of method calls, it prevents other parts of

Canvas from having to understand how messages are displayed.

Canvas supports the use of semaphores by supplying the methods add–semaphore, check–semaphore, lock–semaphore, and release–semaphore. This allows for new to be easily added. Currently, two semaphores exist: mode and popup. The mode semaphore is locked when a user is in the middle of using one of the tools. The popup semaphore is locked when a popup dialog is currently up. If either semaphore is locked, many components of Canvas (such as the menu items) are disabled.

The method set–canvas–property–button is used to identify the button used to change a drawing sheet property. This was necessary, since these buttons are supposed to display the current value of the property. When a drawing sheet property is changed, the button is automatically updated by Canvas.

Finally, the methods add–application–property, get–application–property, and change–application–property are used to manipulate the properties of the Canvas application. These are properties that effect the interaction with the user, such as whether snap coordinates are used, what the grid size is, and whether rulers are displayed. Of course, properties added are only observed by objects programmed to be aware of the properties.

Chapter 6

The Files

There are two types of files generated by Canvas: Canvas files and Curl files. A Canvas file is used to save state between sessions. A Curl file is used to export the Canvas contents for use in a Curl web page.

A separate format was chosen for Canvas files for three reasons. First, the amount of time required for reading and writing files is decreased when using an optimal format for the requirements of Canvas, and the Canvas files may take advantage of the most efficient storage of its information. Second, it is easier, as a programmer, to be able to write and read any format that you wish, rather than a format that must also be correct Curl code. This is important since Canvas is meant to be extended by its users, and any new object types added to Canvas will have to save and load their internal state. Third, if Canvas files were written in Curl code, the users might edit these files. Once a file is edited, it might not be possible for Canvas to interpret that file, and the users might be confused as to the reason why.

Each type of file is created by selecting the appropriate menu option from the Canvas menu. In the rest of this chapter, we briefly discuss about each type of file.

The internal files are created by a two step process. First, the current file format version is written as a comment in the file specified by the user. This version number allows multiple saving formats to be supported in future versions of Canvas. For the second step, each object in the drawing sheet has its save–to–file method called. Objects are given the power to save in any format that they wish, and the application

is kept free from any knowledge of this format.

When exporting Curl files, the process is again two steps. In the first step, all NewCanvasObjects that are also considered objects in Curl are exported using the syntax for defining a Curl class. Currently, no such objects exists, but in future versions this would include Tables, groups of objects, and formatted text. In the second step, a class is written that represents the entire drawing sheet. This class will inherit from Frame (see Chapter 2). All items already saved as classes will be added to and placed in the Frame as Curl objects. All other objects will have their export–draw method called, so as to place their output in the draw method of the Frame.

Chapter 7

Future Work

As stated in the introduction, this thesis created the framework for Canvas; the current state of Canvas is not a complete application. In this chapter, we list a few items that are currently missing in Canvas. In addition, we discuss features found to be currently missing in Curl that make some Canvas features impossible. However, we note that since Curl is in its Alpha stages, missing features are to be to be expected.

7.1 Canvas

Currently, there are many properties added to Canvas that have no effect. For example, although a property for grid size exists, currently it is not used. All added properties need to be made effective throughout the code.

Secondly, a number of common drawing mechanisms are not yet implemented. Drawing modes should include polygons, circles, and freehand drawing. Objects also need a way to draw arrows, dashed lines, and fill patterns. Needed edit modes include moving vertices and editing with a dialog box. Needed actions include align, flip, and distribute.

In general, many dialog boxes are needed. Currently, the only way to change many of Canvas's properties is by programming a call to change–canvas–property or change–application–property. Also, the drawing modes do not currently have a way to enter points through a dialog. Finally, the dialog boxes currently implemented do

not offer an optimal graphical user interface, although they are sufficient.

Canvas needs more functionality added to it. Needed toolbars include one for Tables, one for text, and one for forms.

Finally, Canvas needs more testing. Additions were being made to Canvas until the end. Although each addition was tested, the thoroughness of the testing was not sufficient.

7.2 Curl

The drawing mechanism in Curl is incomplete. The most needed feature for Canvas is the ability to draw an arc. Without this feature, curved rectangles, rotated ellipses, and all arcs will not be possible. A more complex problem is the fact that an object of type GraphicContext (and not a subclass of GraphicContext) is the only class that may be sent to the draw method. Therefore, users of Curl are limited to the methods available in GraphicContext. The ability to copy and subclass the GraphicContext, and then pass this new GraphicContext, is needed in Curl.

Drawing xor lines is currently a limited feature in Curl. First, the current drawing protocol does not allow for xor lines to be used during the draw method, and regular lines cannot be drawn outside of the draw method. This prevents Canvas from having consistent drawing during the creation of a new object. Second, objects are not notified when the Curl browser is iconified. For this reason, it is impossible to tell if the xor lines are still valid or not.

The current mechanism for event handling in Curl is good; however, two features are missing. The first is the ability to ignore mouse clicks. With this ability, a Curl object would be able to pretend, for the moment, to not be an event handler, and the mouse event would go to the next–most–reasonable object. Without this feature, it is either impossible for a new polyline to have a vertex inside an established rectangle, or it is impossible for an object to respond to mouse events for selection or movement. To get around this missing feature, Canvas currently duplicates much of the event handling mechanism. The second missing feature for event handling in

Curl is the ability to simulate new events, including mouse movement, mouse and keyboard presses, and dragging. Just as it is easier for the user to create a web page using a visual tool, it is easier for the components of Canvas to communication with other components by simulating pointer and keyboard events.

Chapter 8

Conclusions

Canvas was written with three goals in mind. First, it was created to reduce the learning curve for Curl. Second, Canvas was meant to facilitate quick creation of Curl pages. Canvas accomplishes these first two tasks by removing the user from most of the code. However, the currently limited number of features of Canvas minimizes its advantages. Finally, Canvas was meant to be extended easily. Currently, many mechanisms are in place for functionality to be added to Canvas. However, a trade–off was made between easy creation of new components and the functionality of Canvas. As objects in Canvas acquire more functionality, the time to create such an object will unfortunately increase.

In short, although much work remains to be done, it is hoped that the work presented in this thesis provides the foundation for creating a diverse Curl development environment.

Appendix A

Code

A.1 Generic Tools

A.1.1 common.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| These are things that I think should be defined for all users of
| Curl.

{define {round x:float}:int
  {if {< x 0}
      {return {- x .5}}
      {return {+ x .5}}
      }}

{define {->text thing:any}:text
  {return {format void "~p" thing}}
  } | procedure: ->text

{define {text-width t:text g:Graphic start:int=0 len:int=0}:int
  | Find the width of the text string using the default font for the
  | graphic.  If length is 0, assumes the full text string.

  font:FFont
  gc:GraphicContext

  {set font {get-font g}}
  {if {= len 0}
      {set len {- {length t} start}}
```

```
    }
  {return {TextWidth font t start len}}
  } | procedure: text-width

{define {text-height g:Graphic}:int
  {return {get-font g}.pointsize}
  } | procedure: text-height

{define {myapply-method class:type slot:symbol obj:any l:list}
  {apply-method class slot {list->vector {cons obj l}}}
  } | procedure: myapply-method

{define-constant MAXINT:int=#x7FFFFFFF}

{define {bit-not i:int}:int
  {return {bit-xor #xFFFFFFFF i}}}

{define public {rubberband-lines i:Graphic pts:list}:void
  len:int={list-length pts}
  point1:Point
  point2:Point
  {cond {{= 0 len} {return}}
        {{= 1 len}
         {set point1 {car pts}}
         {set point2 {new Point point1.x {+ 1 point1.y}}}
         {rubberband-line i point1.x point1.y point2.x point2.y}
         }
        {else
         {set point2 {car pts}}
         {set pts {cdr pts}}
         {while {not {void? pts}}
           {set point1 point2}
           {set point2 {car pts}}
           {set pts {cdr pts}}
           {rubberband-line i point1.x point1.y point2.x point2.y}
           } | while
         }} | else-cond
  } | rubberband-lines



||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||


{define {type->symbol t:any}:symbol
```

```
  | hack: nasty hack to get around permission problems.
  sb:StringBuffer={get-string-buffer}
  txt:text
  {if {void? t} {return void}}
  {format sb "~p" t}
  {dotimes {i 6}
    {sb.ReadCh}}
  {set t {read-string sb}}
  {return {text->symbol t}}
  } | procedure: type->symbol

{define {char->int ch:char return-negative:bool=true}:int
  {cond {{and {>= ch #\0} {<= ch #\9}}
         {return {- ch #\0}}
         }
        {{and {>= ch #\A} {<= ch #\Z}}
         {return {+ {- ch #\A} 10}}
         }
        {{and {>= ch #\a} {<= ch #\z}}
         {return {+ {- ch #\a} 10}}
         }
        {{< ch 0}
         {return ch}
         }
        {else
         {if return-negative
             {return -1}
             {return 0}
             }
         }} | else and cond
  } | procedure: char->int


{define {text->int t:text start:int=0 base:int=10
                   return-void:bool=false return-length:bool=false}:any
  | returns the int and the number of characters used to make that int
  i:int
  len:int
  ans:any
  read-proc:{proc {}:char}
  peek-proc:{proc {}:char}

  {set i start}
  {set len {length t}}

  {set read-proc
       {lambda {}:char
```

```
        {if {>= i len}
            {return -1}}
        {set i {+ i 1}}
        {return {aref t {- i 1}}}
        }}
  {set peek-proc
       {lambda {}:char
        {if {>= i len}
            {return -1}}
        {return {aref t i}}
        }}

  {set ans {generic-parse-int peek-proc read-proc base=base}}
  {if {void? ans}
      {if return-void
          {return void}
          {if return-length
              {return {make-list 0 0}}
              {return 0}}}
      {if return-length
          {return {make-list ans {- i start}}}
          {return ans}
          }
      }
  } | procedure: text->int


{define {text->float t:text start:int=0 base:int=10
                     return-void:bool=false return-length:bool=false}:any
  i:int
  len:int
  ans:any
  read-proc:{proc {}:char}
  peek-proc:{proc {}:char}

  {set i start}
  {set len {length t}}

  {set read-proc
       {lambda {}:char
        {if {>= i len}
            {return -1}}
        {set i {+ i 1}}
        {return {aref t {- i 1}}}
        }}
  {set peek-proc
       {lambda {}:char
```

```
        {if {>= i len}
            {return -1}}
        {return {aref t i}}
        }}

  {set ans {generic-parse-float peek-proc read-proc base=base}}
  {if {void? ans}
      {if return-void
          {return void}
          {if return-length
              {return {make-list 0 0}}
              {return 0}}}
      {if return-length
          {return {make-list ans {- i start}}}
          {return ans}}
      }
  } | procedure: text->float


{define private {generic-parse-int peek-char-proc:{proc {}:char}
                                   read-char-proc:{proc {}:char}
                                   base:int=10}:any

  ch:char
  chVal:int
  sum:int
  negative:bool

  {loop
    {set ch {peek-char-proc}}
    {if {< ch 0}
        | no integer here
        {return void}
        }
    {if {not {isspace ch}}
        {break}
        {read-char-proc}
        }
    } | loop
  | do we have a negative sign?
  {when {= ch #\-}
    {set negative true}
    {read-char-proc}
    {set ch {peek-char-proc}}
    } | when
  | do we have a number?
  {set chVal {char->int ch}}
```

```
  {if {or {< chVal 0} {>= chVal base}}
      {return void}
      }

  | ALRIGHT!  We got something useful!
  {set sum chVal}
  {read-char-proc}
  {loop
    {set ch {peek-char-proc}}
    {set chVal {char->int ch}}
    {if {or {>= chVal base} {< chVal 0}}
        {break}}
    {set sum {+ {* sum base} chVal}}
    {read-char-proc}
    } | while

  {if negative
      {return {- sum}}
      {return sum}
      }
  } | procedure generic-parse-int


{define private {generic-parse-float peek-char-proc:{proc {}:char}
                                     read-char-proc:{proc {}:char}
                                     base:int=base}:any
  sum:float
  temp-num:any
  decimal-read-proc:{proc {}:char}
  decimal-peek-proc:{proc {}:char}
  i:int

  {set temp-num
       {generic-parse-int peek-char-proc read-char-proc base=base}}
  {if {void? temp-num}
      {return void}
      {set sum temp-num}}
  | Ok, are we at a decimal place?
  {if {not {= {peek-char-proc} #\.}}
      {return sum}
      {read-char-proc}
      }
  | OK, make sure the next thing is a digit
  {set temp-num {char->int {peek-char-proc}}}
  {if {or {< temp-num 0} {>= temp-num base}}
      {return sum}}
```

```
  {set decimal-read-proc
       {lambda {}:char
         {set i {+ i 1}}
         {return {read-char-proc}}}}
  {set decimal-peek-proc peek-char-proc}
  {set temp-num {generic-parse-int decimal-peek-proc decimal-read-proc
                                   base=base}}
  {if {< sum 0}
      {return {- sum {/ temp-num {power base i}}}}
      {return {+ sum {/ temp-num {power base i}}}}
      }
  } | generic-parse-float


{define {read-int fin:InputPort base:int=10}:any
  read-proc:{proc {}:char}
  peek-proc:{proc {}:char}

  {set read-proc {lambda {}:char {return {fin.ReadCh}}}}
  {set peek-proc {lambda {}:char {return {fin.PeekCh}}}}

  {return {generic-parse-int peek-proc read-proc base=base}}
  } | procedure: read-int


{define {read-float fin:InputPort base=10}:any
  read-proc:{proc {}:char}
  peek-proc:{proc {}:char}

  {set read-proc {lambda {}:char {return {fin.ReadCh}}}}
  {set peek-proc {lambda {}:char {return {fin.PeekCh}}}}

  {return {generic-parse-float peek-proc read-proc base=base}}
  } | procedure: read-float


{define {read-line fin:InputPort}:text
  sb:StringBuffer={get-string-buffer}
  ch:char

  {if {< {fin.PeekCh} 0}
      {return void}}
  {loop
    {set ch {fin.ReadCh}}
    {if {or {= ch #\NEWLINE} {< ch 0}}
        {return {sb.text}}
        {sb.WriteCh ch}}
```

```
    }
  } | procedure read-line


{define {read-string fin:InputPort}:text
  sb:StringBuffer={get-string-buffer}
  ch:char
  | skip the white stuff
  {loop
    {set ch {fin.PeekCh}}
    {if {< ch 0}
        {return void}}
    {if {not {isspace ch}}
        {break}
        {fin.ReadCh}}
    } | loop
  | get the string
  {loop
   {set ch {fin.ReadCh}}
    {if {< ch 0}
        {return void}}
    {if {isspace ch}
        {return {sb.text}}
        {sb.WriteCh ch}}
    }
  } | method: read-string


{define {format-in fin:InputPort control:text}:list
  | takes in a control list and output the right number of args.  From
  | there, the user shoulw do something like:
  | It's not great, but it's all that I can do without pointers.
  return-list:list
  return-item:any
  i:int
  len:int={length control}
  c:char

  {if {void? fin}
      {error "format-in sent void InputPort"}
      }

  | bugbug: should be a tagged-loop.  But that's currently not working
  {loop
    {if {>= i len} {break}}
    {set c {aref control i}}
    {cond {{!= c #\~}
```

```
        {if {not {isspace c}}
            {error "Invalid control string passed to format-in"}
            {set i {+ i 1}}
            }
        }
       {else
        | we're doing a ~ thing.
        {set c {aref control {+ i 1}}}
        {if {and {>= c #\A} {<= c #\Z}} {set c {+ c 32}}}
        {cond {{= c #\%}
               | read until a newline and then return string
               {set return-item {read-line fin}}
               }
              {{= c #\s}
               {set return-item {read-string fin}}
               }
              {{= c #\d}
               {set return-item {read-int fin}}
               }
              {{= c #\o}
               {set return-item {read-int fin base=8}}
               }
              {{= c #\x}
               {set return-item {read-int fin base=16}}
               }
              {{= c #\b}
               {set return-item {read-int fin base=16}}
               }
              {{= c #\f}
               {set return-item {read-float fin}}
               }
              {else
               {error "Bad format control " c}
               }
              } | inner cond

        {when {void? return-item}
          {break}
          }
        {set return-list {cons return-item return-list}}
        {set i {+ i 2}}
        } | else
       } | outter-cond
  } | loop

| Ok, we made it, (although maybe not completely), we have our list,
| it's just in the wrong order.
```

```
  {return {reverse return-list}}
  } | procedure: format-in
```

A.1.2 generic.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| These are handy generic things that many users might want, but not
| generic enough that I would want to clutter the namespace with
| them.  A couple of classes here are hacks for mixin-like reasons.

{define-constant PI:float=3.14159265358979323846}


{define {average x:float y:float}:float
  {return {/ {+ x y} 2}}}
{define {fabs x:float}:float
  {if {< x 0}
      {return {- x}}
      {return x}}}
{define {square x} {return {* x x}}}

{define {sqrt num:float}:float
  old-guess:float
  new-guess:float

  {set old-guess 1}
  {dotimes {i 100}
    {set new-guess {/ {+ old-guess {/ num old-guess}} 2}}
    {if {< {/ {fabs {- new-guess old-guess}} new-guess} 0.0001}
        {return new-guess}
        {set old-guess new-guess}}
    }
  {return old-guess}
  } | procedure: sqrt




{define {generic-apply-method class:type slot:symbol obj:any l:list ...}
  {myapply-method class slot obj {append {rest-as-list} l}}}


{define {centering-offset obj:any
                          container:Graphic=void
                          horizontal:bool=true}:int
```

```
  {cond {{isa Graphic obj}
         {if horizontal
             {return {/ {obj.graphic-get-width} 2}}
             {return {/ {obj.graphic-get-height} 2}}
             }
         {obj.graphic-get-width}
         }
        {{isa text obj}
         {if {void? container}
             {error {format void "~p~p"
                            "Cannot compute centering offset "
                            "of text without a container"}}
             }
         {if horizontal
             {return {/ {text-width obj container} 2}}
             {return {/ {text-height container} 2}}
             }
         }
        {else
         {error "Cannot compute centering offset: unknown type"}
         }
        } | cond
  } | procedure: centering-offset

{define public {filter p:{proc {any}:bool} l:list}:list
  {until {or {void? l} {p l.first}}
    {set l l.rest}}
  {when {void? l}
      {return void}}
  | We guarentee a list to be returned
  {let result:list={alloc list}
       cur-pos:list=result
    {set result.first l.first}
    {set l l.rest}
    {loop
      {until {or {void? l} {p l.first}}
        {set l l.rest}}
      {if {void? l}
          {return result}}
      {set cur-pos.rest {alloc list}}
      {set cur-pos cur-pos.rest}
      {set cur-pos.first l.first}
      {set l l.rest}
      }}} | filter

{define public {memq-by-proc p:{proc {any}:bool} l:list}:list
  {until {void? l}
```

```
    {if {p l.first} {return l}}
    {set l l.rest}}
  {return void}}

{define public {cdr-down-by-proc p:{proc {list}:bool} l:list}:list
  {until {void? l}
    {if {p l} {return l}}
    {set l l.rest}}
  {return void}}

{define {color->text color:any quote-sym:bool=false}:text
  {if {isa symbol color}
      {if quote-sym
          {return {text-append "'" {->text color}}}
          {return {->text color}}}}
  {set color {any-to-color color}}
  {if {void? color}
      {return {->text void}}}
  {return {format void "{FColor ~d ~d ~d}" color.r color.g color.b}}
  } | color->text




{define-class Point {}
  | Essentially, this is a data structure, for easy passing and
  | some efficiency.
  x:float
  y:float
  int-numbers:bool

  {define {init x:float y:float int-numbers:bool=true}
    {cond {int-numbers
           {set self.x {cast int {round x}}}
           {set self.y {cast int {round y}}}
           }
          {else
           {set self.x x}
           {set self.y y}
           }
          }
    {set self.int-numbers int-numbers}
    } | method: init
  {define {trace}
    {output {self.return-text}}
    }
  {define {return-text}:any
```

```
    {return {format void "(~p, ~p)" {->text self.x} {->text self.y}}}
    }
  {define {copy}:Point
    {return {new Point self.x self.y}}
    } | method: copy
  {define {same-as p:Point tolerance:float=.0001}:bool
    {return {and {< {fabs {- p.x self.x}} tolerance}
                 {< {fabs {- p.y self.y}} tolerance}}}
    } | method: same-as
  } | class: Point


{define-class public CanvasPropertyList {PropertyList}
  | This class exists so I can use PropertyList as a
  | (member-variable) type.  In addition, I want to be able to list
  | all properties.
  {define public {init}
    {return}}

  {define public {get-all-properties}:Property
    {return self.properties}
    } | method: get-all-properties

  {define public {get-all-property-names}:list
    temp-list:list
    temp-property:Property
    {set temp-property self.properties}
    {while {not {void? temp-property}}
      {set temp-list {cons temp-property.name temp-list}}
      {set temp-property temp-property.next}
      }
    {return temp-list}
    } | method: get-all-property-names

  {define public {get-all-property-values}:list
    temp-list:list
    temp-property:Property
    {set temp-property self.properties}
    {while {not {void? temp-property}}
      {set temp-list {cons temp-property.value temp-list}}
      {set temp-property temp-property.next}
      }
    {return temp-list}
    } | method: get-all-property-values
  } | class: CanvasPropertyList
```

```
{define-class CanvasPointerEvent {PointerEvent}
  {define public {init}
    {return}
    } | method: init
  } | class: CanvasPointerEvent


{define-class CanvasKeyEvent {KeyEvent}
  {define public {init}
    {return}
    } | method: init
  } | class: CanvasKeyEvent

{define-class CanvasDragee {Dragee}
  {define {init}
    {super.init}}
  } | class: CanvasDragee

{define-class ImmitatingGroup {}
  immitating-type:type
  immitating-objects:list  | of "ImmitatingObject"s

  {define {init immitating-type:type}
    {set self.immitating-type immitating-type}
    } | method: init

  {define {add-immitator obj:ImmitatingObject}:int
    {if {and {isa ImmitatingObject obj}
             {void? {memq obj self.immitating-objects}}}
        {cons obj self.immitating-objects}}
    {return {list-length self.immitating-objects}}
    } | method: add-immitator

  {define {delete-immitator obj:ImmitatingObject}:int
    {if {not {void? {memq obj self.immitating-objects}}}
        {set self.immitating-objects {delq! obj self.immitating-objects}}
        }
    {return {list-length self.immitating-objects}}
    } | method: delete-immitator

  {define {propagate-change mimic-proc:{proc {ImmitatingObject}:void}}:void
    {map {lambda {x}  | where x is an ImmitatingObject
           {mimic-proc x}
           }
         self.immitating-objects}
    } | method: propagate-change
  } | class: ImmitatingGroup
```

```
{define-mixin-class ImmitatingObject {}
  group:ImmitatingGroup

  {define {init}
    {return}
    } | method: init

  {define {add-to-immitating-group newgroup:ImmitatingGroup}:void
    {if {not {void? self.group}}
        | need to remove yourself from the previous one.
        {self.group.delete-immitator self}}
    {set self.group newgroup}
    {if {not {void? newgroup}}
        {newgroup.add-immitator self}
        }
    } | method: add-to-immitating-group
  } | class ImmitatingObject


{define-class Stack {}
  stack:list

  {define {init}
    {return}
    }
  {define {peek}:any
    {if {void? self.stack}
        {error "Trying to peek down an empty stack"}
        }
    {return {car self.stack}}
    }
  {define {pop}:any
    temp-guy:any
    {if {void? self.stack}
        {error "Trying to pop an empty stack"}
        }
    {set temp-guy {car self.stack}}
    {set self.stack {cdr self.stack}}
    {return temp-guy}
    }
  {define {push item:any}:void
    {set self.stack {cons item self.stack}}
    }
  {define {empty?}:bool
    {return {void? self.stack}}
```

```
    }
  {define {clear}:void
    {set self.stack void}
    }
  } | class: Stack


{define-class UndoBuffer {}
  items:{array list}
  private buffer-size:int
  private used-buffer:int
  private redo-size:int
  private cur-pos:int
  temp-buffer:list

  {define {init size:int}
    {set self.items {new {array list} size}}
    {set self.buffer-size size}
    } | method: init

  {define {add-item item:any commit:bool=true}:void
    {set self.redo-size 0}
    {set self.temp-buffer {cons item self.temp-buffer}}
    {when {not commit}
      {return}
      }
    {self.commit}
    } | method: add-item

  {define {add-items items:list commit:bool=true}:void
    {dotimes {i {list-length items}}
      {self.add-item {car items} commit=false}
      {set items {cdr items}}
      }
    {self.commit}
    } | method: add-items

 {define {commit}:void
    | the following does the right thing even if we are at max capacity
    {if {void? self.temp-buffer} {return}}
    {set {aref self.items self.cur-pos} self.temp-buffer}
    {set self.cur-pos {% {+ self.cur-pos 1} self.buffer-size}}
    {if {< self.used-buffer self.buffer-size}
        {set self.used-buffer {+ self.used-buffer 1}}
        }
    {set self.temp-buffer void}
    } | method: commit
```

```
  {define {get-undo-items}:list
    temp-list:list

    {when {not {void? self.temp-buffer}}
      | undo-ing in the middle.  Assume that that means they were done
      {set temp-list self.temp-buffer}
      {set self.temp-buffer void}
      {self.add-items temp-list}
      }
    {if {= self.used-buffer 0}
        {return void}
        }
    | hack: need to add the buffersize because {% -1 10} return -1.
    {set self.cur-pos {% {+ {- self.cur-pos 1} self.buffer-size}
                         self.buffer-size}}
    {set temp-list {aref self.items self.cur-pos}}
    {set self.used-buffer {- self.used-buffer 1}}
    {set self.redo-size {+ self.redo-size 1}}
    {return temp-list}
    } | method: get-undo-items

  {define {get-redo-items}:list
    temp-list:list
    {if {= self.redo-size 0}
        {return void}
        }
    {set self.used-buffer {+ self.used-buffer 1}}
    {set self.redo-size {- self.redo-size 1}}
    {set temp-list {aref self.items self.cur-pos}}
    {set self.cur-pos {% {+ self.cur-pos 1} self.buffer-size}}
    {return temp-list}
    } | method: get-redo-items
  } | class: UndoBuffer
```

A.1.3 dynamic.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains the dynamic variables found in the Canvas.

{define-class GraphicDynamic {Frame}
  obj:Dynamic

  {define {init obj:any ...}
    {set self.obj {new Dynamic obj}}
```

```
    | hack: apply-method doesn't work for some reason with Dynamics.
    | Have to do it in two steps.
    {myapply-method Frame 'init self {rest-as-list}}
    {super.append-object self.obj}
    }
  {define {get-value}:any
    {return {self.obj.get-value}}
    }
  {define {set-value val:any}:void
    {self.obj.set-value val}
    }
  } | class: GraphicDynamic


{define-class DynamicStructure {}
  names:list
  values:list
  current-name:symbol

  {define {init}
    {set self.current-name void}
    } | method: init

  {define private {internal-memq name:symbol both-lists:bool=false}:list
    temp-names:list=self.names
    temp-info:list=self.values
    {until {void? temp-names}
      {if {equal? {car temp-names} name}
          {if both-lists
              {return {make-list temp-names temp-info}}
              {return temp-info}
              }
          }
      {set temp-names {cdr temp-names}}
      {set temp-info {cdr temp-info}}
      } | until
    {throw {new Exception void}}
    } | method: internal-memq

  {define {get-value name:symbol}:any
    ans:any
    {try
     {set ans {car {self.internal-memq name}}}
     {catch {e:Exception}
            {error "DynamicStructure.get-value called with unknown name."}
            }
     } | try
```

```
  {return ans}
  } | method: get-value

{define {get-current-value}:any
  {return {self.get-value self.current-name}}}

{define {get-current-name}:symbol
  {return self.current-name}}

{define {set-current-name name:symbol}:void
  {if {void? {memq name self.names}}
      {error "Setting as current name unknown name."}
      {set self.current-name name}}
  } | method: set-current-name

{define {change-value name:symbol new-info:any}:void
  {try
   {set {self.internal-memq name}.first new-info}
   {catch {e:Exception}
          {error {format void "~p~p"
                         "DynamicStructure.change-value "
                         "called with unknown name."}}
          }
   } | try
  } | method: change-value

{define {add-name name:symbol
                  value:any=void
                  add-as-current:bool=true}:void
  {if {not {void? {memq name self.names}}}
      {error {format void "~p~p"
                     "DynamicStructure.add-name "
                     "called with an already known name."}}}
  {set self.names {cons name self.names}}
  {set self.values {cons value self.values}}
  {if add-as-current
      {set self.current-name name}
      }
  } | add-name

{define {remove-name name:symbol}:void
  temp-names:list=self.names
  temp-info:list=self.values
  {if {void? {memq name self.names}}
      {error "Remove-name called with an unknown name."}}
  {if {equal? self.current-name name}
      {set self.current-name void}}
```

```
    {when {equal? name {car self.names}}
      {set self.names {cdr self.names}}
      {set self.values {cdr self.values}}
      {return}
      } | when
    {until {equal? {cadr temp-names} name}
      {set temp-names {cdr temp-names}}
      {set temp-info {cdr temp-info}}
      }
    {set temp-names.rest {cdr temp-names}.rest}
    {set temp-info.rest {cdr temp-info}.rest}
    } | remove-name

  {define {is-name? name:symbol}:bool
    {return {not {void? {memq name self.names}}}}
    } | method: is-name?

  {define {get-name value:any}:symbol
    temp-names:list=self.names
    temp-values:list=self.values
    {until {void? temp-values}
      {if {equal? {car temp-values} value}
          {return {car temp-names}}
          }
      {set temp-names {cdr temp-names}}
      {set temp-values {cdr temp-values}}
      } | until
    } | get-name

  {define {list-names}:list
    {return self.names}
    } | list-names
  } | class: DynamicStructure


{define-class DynamicRotateStructure {DynamicStructure}
  names-in-order:list

  {define {init}
    {super.init}
    } | init

  {define {set-current-name name:symbol}:void
    {super.set-current-name name}
    {if {equal? {car self.names-in-order} name}
        {return}}
    {set self.names-in-order {delq! name self.names-in-order}}
```

```
    {set self.names-in-order {cons name self.names-in-order}}
    } | method: set-current-name

  {define {add-name name:symbol
                   value:any=void
                   add-as-current:bool=true}:void
    last-elt:list
    {super.add-name name value=value add-as-current=add-as-current}

    {when {or add-as-current
              {void? self.names-in-order}}
      {set self.names-in-order {cons name void}}
      {return}
      } | when
    {set last-elt {lastcdr self.names-in-order}}
    {set last-elt.rest {new list name void}}
    } | method: add-name

  {define {remove-name name:symbol}:void
    {super.remove-name name}
    {set self.names-in-order {delq! name self.names-in-order}}
    } | method: remove-name
  } | class DynamicRotateStructure



{define-class DynamicMenu {GraphicDynamic DynamicStructure}
  {define {init init-name:symbol init-menu:MenuBar={new MenuBar}}
    {invoke-method GraphicDynamic 'init self init-menu}
    {invoke-method DynamicStructure 'init self}
    {self.add-menu init-name}
    {invoke-method DynamicStructure 'set-current-name self init-name}
    } | method: init

  {define {change-dynamic-menu name:symbol}:void
    {invoke-method DynamicStructure 'set-current-name self name}
    {self.update-current-menu}
    } | method: change-dynamic-menu

  {define {update-current-menu}:void
    new-value:MenuBar
    {set new-value
         {{invoke-method
              DynamicStructure
              'get-current-value self}.make-MenuBar}}
    {invoke-method GraphicDynamic 'set-value self new-value}
    } | method: update-current-menu
```

```
{define {add-menu name:symbol value:ListMenuBar={new ListMenuBar}}:void
  {invoke-method DynamicStructure 'add-name self name value=value}
  } | method: add-menu

{define {add-menu-action path:list
                         name:any
                         before:any=void after:any=void
                         action:action-type=no-action
                         menu-name:symbol=self.current-name}:void
  submenu:ListSubMenu
  {set submenu
       {{invoke-method
            DynamicStructure
            'get-current-value self}.return-submenu path}}
  {if {void? submenu}
      {error "Adding a MenuAction to an invalid path"}
      }
  {submenu.insert-object {new ListMenuAction name action}
                         before=before after=after}
  | just mutated the value, so don't need to set a new value.  Do
  | need to update the menu, however.
  {if {equal? {invoke-method DynamicStructure 'get-current-name self}
              menu-name}
      {self.update-current-menu}
      }
  } | method: add-menu-action


{define {remove-menu-action path:list
                            name:any
                            menu-name:symbol=self.current-name}:void
  submenu:ListSubMenu
  menu-items:list

  {set submenu
       {{invoke-method
            DynamicStructure
            'get-current-value self}.return-submenu path}}
  {if {void? submenu}
      {error "Removing a MenuAction from an invalid path"}
      }

  {if {not {submenu.is-object? name type-of-object=ListMenuAction}}
      {error {format void "Removing unknown MenuAction ~p" name}}
      {submenu.remove-object name}}
```

```
    | just mutated the value, so don't need to set a new value.  Do
    | need to update the menu, however.
    {if {equal? {invoke-method DynamicStructure 'get-current-name self}
                menu-name}
        {self.update-current-menu}
        }
    } | method: remove-menu-action


  {define {add-sub-menu path:list name:any
                        before:any=void after:any=void
                        menu-name:symbol=self.current-name}:void
    submenu:ListSubMenu
    {set submenu
         {{invoke-method
              DynamicStructure
              'get-current-value self}.return-submenu path}}
    {if {void? submenu}
        {error "Adding a sub menu to an invalid path"}
        }
    {submenu.insert-object {new ListSubMenu name}
                           before=before after=after}

    | just mutated the value, so don't need to set a new value.  Do
    | need to update the menu, however.
    {if {eq? {self.get-current-name} menu-name}
        {self.update-current-menu}
        }
    } | method: add-sub-menu
  } | class: DynamicMenu



{define-class DynamicToolbar {DynamicStructure GraphicDynamic}

  {define {init init-name:symbol=void init-toolbar:NewToolbar=void}
    {invoke-method GraphicDynamic 'init self init-toolbar}
    {invoke-method DynamicStructure 'init self}
    {when {not {void? init-name}}
      {self.add-toolbar init-name init-toolbar}
      {self.change-dynamic-toolbar init-name}
      }
    } | method: init

  {define {add-toolbar name:symbol
                       value:NewToolbar
                       add-as-current:bool=true}:void
```

```
    {invoke-method DynamicStructure 'add-name self
      name value=value add-as-current=add-as-current}
    {if add-as-current
        {self.update-current-toolbar}
        }
    } | method: add-toolbar

  {define {change-dynamic-toolbar name:symbol}:void
    {invoke-method DynamicStructure 'set-current-name self name}
    {self.update-current-toolbar}
    } | method: change-dynamic-toolbar

  {define {update-current-toolbar}:void
    new-value:NewToolbar
    {set new-value
         {invoke-method DynamicStructure 'get-current-value self}}
    {invoke-method GraphicDynamic 'set-value self new-value}
    } | method: update-current-toolbar

  {define {add-button toolbar-name:symbol b:NewToolbarButton
                      row:int=-1 col:int=-1}:void
    tbar:NewToolbar
    {set tbar
         {invoke-method DynamicStructure 'get-value self toolbar-name}}
    {tbar.add-button b row=row col=col}
    | mutated, so don't need to set new value.
    | don't need to update-current-toolbar, either, since a Graphic is
    | smart.
    {self.update-current-toolbar}
    {return}
    } | method: add-button
  } | DynamicToolbar
```

A.1.4 menu.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| Dynamic Menu stuff.

{define-class CanvasMenuBar {MenuBar}
  {define {init l:list}
  | another hack, since you can't "apply" the keyword new.
    {myapply-method MenuBar 'init self l}
    } | method: init
  } | class: CanvasMenuBar
```

```
{define-class CanvasSubMenu {SubMenu}
  {define public {init l:list}
    {myapply-method SubMenu 'init self l}
    } | method: init
  } | class CanvasSubMenu


{define-class ListMenuItem {}
  label:any

  {define {init label:any}
    {set self.label label}
    } | method: init
  {define {is-label? label:any}:bool
    {return {equal? self.label label}}
    } | method: is-label?
  {define {get-label}:any
    {return self.label}
    } | method: get-label
  } | class: ListMenuItem


{define-class ListMenuAction {ListMenuItem}
  p:action-type

  {define {init label:any p:action-type}
    {super.init label}
    {set self.p p}
    } | method: init

  {define {make-MenuAction}:MenuAction
    {return {new MenuAction label=self.label action=self.p}}
    } | method: make-MenuAction
  } | class: ListMenuAction

{define-class CanvasListMenuAction {ListMenuAction}
  {define {init ...}
    {myapply-method ListMenuAction 'init self {rest-as-list}}
    }

  {define {make-MenuAction}:MenuAction
    {return {new MenuAction label=self.label action=self.p}}
    } | make-MenuAction
  } | class: CanvasListMenuAction


{define-class ListSubMenu {ListMenuItem}
```

```
sub-parts:list

{define {init label:any}
  {set self.label label}
  } | method: init

{define {insert-object o:any before:any=void after:any=void}:void
  | defaults to the end
  temp-list:list

  {if {not {void? {memq-by-proc
                   {lambda {x}:bool
                     {return {equal? o x}}}
                   self.sub-parts}}}
      {throw {new Exception
                  {format void
                          "Inserting an already known menu item ~p"
                          o}}}
      }
  {cond {{not {void? before}}
         {when {{car self.sub-parts}.is-label? before}
           {set self.sub-parts {cons o self.sub-parts}}
           {return}
           }
         {set temp-list self.sub-parts}
         {while {and {not {void? temp-list}}
                     {not {void? {cdr temp-list}}}
                     {not {{cadr temp-list}.is-label? o}}}
           {set temp-list {cdr temp-list}}}
         {if {void? {cdr temp-list}}
             {throw {new Exception
                         {format void "~p~p~p"
                                 "Attempt to insert menu item "
                                 "before unknown item "
                                 before}}}}
         } | cond-not-void-before
        {{not {void? after}}
         {set temp-list
              {memq-by-proc {lambda {x}:bool
                              {return {x.is-label? after}}
                              }
                            self.sub-parts}}
         {if {void? temp-list}
             {throw {new Exception
                         {format void "~p~p~p"
                                     "Attempt to insert menu item "
                                     "after unknown item "
```

```
                                        after}}}}
        } | cond not-void-after
       {else
        {set self.sub-parts {append self.sub-parts {make-list o}}}
        {return}
        }} | else-cond
  | OK, if we're here, temp-list should have the where
  | we want to put this guy.
  {set temp-list.rest {cons o temp-list.rest}}
  } | method: insert-object

{define {get-object name:symbol
                    type-of-object:type=ListMenuItem}:ListMenuItem
  temp-list:list
  {set temp-list {memq-by-proc
                 {lambda {x:any}:bool
                   {if {and {isa type-of-object x} {x.is-label? name}}
                       {return true}
                       {return false}
                       }}
                 self.sub-parts}}
  {if {void? temp-list}
      {return void}
      {return {car temp-list}}
      }
  } | method: get-object

{define {is-object? name:symbol type-of-object:type=ListMenuItem}:bool
  {if {void? {self.get-object name type-of-object=type-of-object}}
      {return false}
      {return true}
      }
  } | method: is-object?

{define {remove-object name:symbol type-of-object:type=ListMenuItem}:void
  object:ListMenuItem
  {set object {self.get-object name type-of-object=type-of-object}}
  {if {void? object}
      {throw {new Exception
                  {format void "~p~p"
                          "Attempt to remove unknown menu item "
                          name}}}
      {set self.sub-parts {delq! object self.sub-parts}}
      }
  } | method: remove-object

{define {is-label? l:any}:bool
```

```
    {return {equal? l self.label}}
    } | method: is-label?

  {define {make-SubMenu}:SubMenu
    l:list
    {set l {map {lambda {x}:any
                  {if {isa ListSubMenu x}
                      {return {x.make-SubMenu}}
                      {return {x.make-MenuAction}}
                      }
                  }
                self.sub-parts}}
    {return {new CanvasSubMenu
                 {append l
                         {make-list label=self.label}}}}
    } | method: make-SubMenu

  {define {return-submenu path:list}:ListSubMenu
    finding-item:any
    current-item:any
    part-menu:list

    {if {void? path}
        {return self}
        }
    {set finding-item {car path}}
    {set path {cdr path}}
    {set part-menu self.sub-parts}
    {while {not {void? part-menu}}
      {set current-item {car part-menu}}
      {set part-menu {cdr part-menu}}
      {if {and {isa ListSubMenu current-item}
               {current-item.is-label? finding-item}}
          {if {void? path}
              {return current-item}
              {return {current-item.return-submenu path}}
              }
          }
      } | while
    | didn't find it
    {return void}
    } | method: return-submenu
  } | class: ListSubMenu


{define-class ListMenuBar {ListSubMenu}
  {define {init}
```

```
    {super.init 'dummy}
    } | method: init

  {define {make-MenuBar}:MenuBar
    l:list
    {set l {map {lambda {x}:any
                  {if {isa ListSubMenu x}
                      {return {x.make-SubMenu}}
                      {return {x.make-MenuAction}}
                      }
                  }
                self.sub-parts}}
    {return {new CanvasMenuBar l}}
    } | method: return-MenuBar
  } | class: ListMenuBar
```

A.1.5 toolbar.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| generic toolbar stuff.  Possibly handy for other Curl users.

{define-class NewToolbarButton {Button}
  | These are basically buttons.
  {define {init label:Graphic=void value:Graphic=void
                action:action-type=no-action}
    {super.init label=label action=action tooltip=value}
    } | method: init
  } | class: ToolbarButton


{define-class NewToolbar {Table}
  | basically, this is either an HBox or VBox of ToolbarButtons
  horizontal:bool
  NumButtons:int
  test:{proc {any}:bool}    | this is an extra test that all items
                            | in table must pass.

  {define {init horizontal:bool=true
                test:{proc {any}:bool}={lambda {x:any}:bool {return true}}
                cell-border-width=0
                border-width=2 spacing=2 cell-margin=2}
    {super.init}
    {self.set-property cell-border-width=cell-border-width
                       border-width=border-width
                       spacing=spacing cell-margin=cell-margin}
```

```
    {set self.horizontal horizontal}
    {set self.test test}
    } | method: init

  {define {add-button button:NewToolbarButton
                      row:int=-1 col:int=-1}:void
    | WARNING: if you don't use default row-col, you may accidentally
    | remove a button that is already in place.  (Or get a button
    | booted out later.)  Use with caution.  Probably always use or
    | never use.
    {cond {{self.test button}
           {if {< row 0}
               {if self.horizontal
                   {set row 0}
                   {set row self.NumButtons}
                   }}
           {if {< col 0}
               {if self.horizontal
                   {set col self.NumButtons}
                   {set col 0}
                   }}
           {super.add-object button row col}
           {set self.NumButtons {+ self.NumButtons 1}}
           }
          {else
           {error "NewToolbar.add-button called with an invalid button."}
           }
          } | cond
    } | method: add-button

  } | class: NewToolbar



{define-class NewRadioToolbarButton {NewToolbarButton}
  | This is somewhat like a Button, except the only state-changing
  | mouse event is a pointer press.
  | It is found with other NewRadioToolbarButtons in a NewRadioToolbar.

  isselected:bool
  toolbar:NewRadioToolbar
  calling-pointer-release-internally:bool

  {define {init label:Graphic=void
                value:any=void action:action-type=no-action
                selected?:bool=false}
    {set self.isselected selected?}
```

```
  {super.init label=label value=value action=action}
  } | method: init

{define {set-toolbar tb:NewRadioToolbar}:void
  {set self.toolbar tb}
  } | method: self.toolbar

{define {selected?}:bool
  {return self.isselected}
  } | method: selected?

{define {unselect}:void
  {set self.isselected false}
  {self.update}
  } | method: unselect

{define {select}:void
  {set self.isselected true}
  {self.update}
  } | method: select

{define public {pointer-press e:PointerEvent}:void
  {self.toolbar.set-active self}
  {super.pointer-press e}
  | hack: want to pass an extra arg to pointer-release
  {set self.calling-pointer-release-internally true}
  {self.pointer-release e}
  {set self.calling-pointer-release-internally false}
  } | method: pointer-press

{define public {pointer-release e:PointerEvent}:void
  {if {not self.calling-pointer-release-internally}
      | hack: so that the thunk doesn't get called twice
      {self.leave e}
      }
  {super.pointer-release e}
  } | method: pointer-release

{define protected {update}:void
  | This is called essentially as "draw"
  | We want it to be sunken no matter where the mouse is, or if the
  | mouse has been released or not.
  {self.set-property cell-border-style={if self.isselected
                                        'sunken
                                        'raised}}
  {self.invalidate}
  } | method: update
```

```
  } | class: NewRadioToolbarButton


{define-class NewRadioToolbar {NewToolbar}
  | May include items that are not NewRadioToolbarButtons.  However,
  | they must be ToolbarButtons.
  | Only one NewRadioToolbarButton may be selected at a time.
  active-button:NewRadioToolbarButton
  default-button:NewRadioToolbarButton

  {define {init horizontal:bool=true
               test:{proc {any}:bool}={lambda {any}:bool {return true}}
               ...}
    {generic-apply-method NewToolbar 'init self
                          {rest-as-list}
                          horizontal=horizontal test=test}
    } | method: init

  {define {set-default-button button:NewRadioToolbarButton}:void
    | assumes that you've added it to here...
    {set self.default-button button}
    } | method: set-default-button

  {define {set-active button:NewRadioToolbarButton}:void
    {if {eq? self.active-button button}
        {return}
        }
    {if {not {void? self.active-button}}
        {self.active-button.unselect}}
    {set self.active-button button}
    {if {not {void? button}}
        {self.active-button.select}
        }
    } | method: set-active

  {define {add-button button:NewToolbarButton
                      row:int=-1 col:int=-1}:void
    | WARNING: if you don't use default row-col, you may accidentally
    | remove a button that is already in place.  (Or get a button
    | booted out later.)  Use with caution.  Probably always use or
    | never use.
    {super.add-button button row=row col=col}
    {when {isa NewRadioToolbarButton button}
      {{cast NewRadioToolbarButton button}.set-toolbar self}
      {if {invoke-method NewRadioToolbarButton 'selected? button}
          {self.set-active button}
```

```
        }
      }
    } | method: add-button

  {define {set-none-selected}:void
    {if {not {void? self.active-button}}
        {self.active-button.unselect}}
    } | method: set-none-selected
  } | class NewRadioToolbar
```

A.1.6 type.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains stuff that is defined in type.h, for C.

{define {isalnum ch:int}:bool
  {return {or {and {>= ch 48} {<= ch 57}}
              {and {>= ch 65} {<= ch 90}}
              {and {>= ch 97} {<= ch 122}}
              }}}

{define {isalpha ch:int}:bool
  {return {or {and {>= ch 65} {<= ch 90}}
              {and {>= ch 97} {<= ch 122}}
              }}}

{define {isascii ch:int}:bool
  {return {and {>= ch 0} {<= ch 127}}
          }}


{define {isblank ch:int}:bool
  {return {or {= ch 9} {= ch 32}
              }}}

{define {iscntrl ch:int}:bool
  {return {or {and {>= ch 0} {<= ch 31}}
              {= ch 127}
              }}}

{define {isdigit ch:int base:int=10}:bool
  {cond {{= base 10}
         {return {and {>= ch 48} {<= ch 57}}}}
        {{or {< base 0} {> base 36}}
         {throw {new Exception
```

```
                    {format void "Unknown representation of base ~d"
                            base}}}
          }
         {else {return {and {>= ch 48} {<= ch 57}
                            {or {and {>= ch 65} {<= ch {+ 65 {- base 10}}}}
                                {and {>= ch 97} {<= ch {+ 97 {- base 10}}}}
                                }}}

               }}}

{define {isgraph ch:int}:bool
  {return {and {>= ch 33} {<= ch 126}
               }}}

{define {islower ch:int}:bool
  {return {and {>= ch 97} {<= ch 122}
               }}}

{define {isprint ch:int}:bool
  {return {and {>= ch 32} {<= ch 126}
               }}}

{define {ispunct ch:int}:bool
  {return {or {and {>= ch 33} {<= ch 47}}
              {and {>= ch 58} {<= ch 64}}
              {and {>= ch 91} {<= ch 96}}
              {and {>= ch 123} {<= ch 126}}
              }}}

{define {isspace ch:int}:bool
  {return {or {and {>= ch 9} {<= ch 13}}
              {= ch 32}
              }}}

{define {isupper ch:int}:bool
  {return {and {>= ch 65} {<= ch 90}
               }}}
```

A.2 Non–Visual Components

A.2.1 cevhand.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| Defined in here are mixin classes only.  The mixin classes are
```

```
| pseudo event handlers for the canvas.  Subclasses are currently in
| their own files.

{define-mixin-class NewCanvasEventHandler {}
  canvas:NewCanvasApplication
  invalidated:bool

  {define {init}
    {return}
    } | method: init
  | the following are the same as those passed to an EventHandler.
  | The Event passed with both regular coords and snap coords.
  {define {pointer-motion e:PointerEvent snap-point:Point}:void
    {return}}
  {define {pointer-release e:PointerEvent snap-point:Point}:void
    {return}}
  {define {pointer-press e:PointerEvent snap-point:Point}:void
    {return}}
  {define {pointer-dbl-clk e:PointerEvent snap-point:Point}:void
    {return}}
  {define {key-press e:KeyEvent}:void
    {return}}
  {define {esc-key-press}:void
    {self.cancel}}
  {define {cancel}:void
    {return}}
  {define {draw gc:GraphicContext}:void
    {set self.invalidated false}}
  {define {selected}:void
    {return}}
  {define {set-canvas canvas:NewCanvasApplication}:void
    {set self.canvas canvas}}
  {define {invalidate}:void
    {set self.invalidated true}}
  {define {popup-dialog-box}:void
    {self.canvas.canvas.Sheet.error
     "Cannot popup a dialog for this mode."}}
  } | mixin-class: NewCanvasEventHandler


{define-mixin-class NewCanvasDrawMode {NewCanvasEventHandler}
  } | mixin: NewCanvasDrawMode

{define-mixin-class NewCanvasEditMode {NewCanvasEventHandler}
  {define {set-selected-objects l:list}:void
    {error "Failure to overwrite NewCanvasEditMode.set-selected-objects"}
    } | method: set-selected-objects
```

```
  } | mixin: NewCanvasEditMode
```

A.2.2 cmenu.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains the procedures called by menu items.

{define {clear-all-menu-action canvas:NewCanvasSheet}:void
  {if {void? canvas.Objects}
      {canvas.error "Nothing to clear"}
      {canvas.exchange-canvas-objects canvas.Objects void commit=true}
      }
  } | clear-all-menu-action


{define {undo-menu-action canvas:NewCanvasSheet}:void
  {canvas.undo}
  } | undo-menu-action


{define {redo-menu-action canvas:NewCanvasSheet}:void
  {canvas.redo}
  } | redo-menu-action


{define {new-file-menu-action canvas:NewCanvasSheet}:void
  app:NewCanvasApplication
  sheet:NewCanvasSheet
  name:symbol
  temp-list:list

  {set app canvas.Container.Container}
  {set temp-list {app.make-new-sheet}}
  } | new-file-menu-action


{define {export-menu-action canvas:NewCanvasSheet}:void
  group-number:int=1
  group-objects:list
  filename:text
  urn:File
  fout:FileOutputPort

  | get the file ready.
  {set filename {popup-text-query "Please enter URN to export to: "}}
```

```
  {if {void? filename}
      {return}}
  {set urn {xresolve-filename filename}}
  {set fout {urn.write-open}}

  {format fout "| File created by Curl's Canvas Application~%~%"}
  | get all the group stuff
  | ck: not currently done, since groups aren't done

  | get the final canvas thing
  {format fout "~%~%| And now everything drawn in the Canvas~%~%"}
  {format fout
          "{define-class class-number-~d {SimpleGraphic}~%"
          group-number}
  {format fout "   width:int~%"}
  {format fout "   height:int~%~%"}
  {format fout "  {define {init}~%"}
  {format fout "    {set self.width ~d}~%" {canvas.graphic-get-width}}
  {format fout "    {set self.height ~d}~%" {canvas.graphic-get-height}}
  {format fout "    {super.init}~%"}
  {format fout "    } | method: init~%~%"}
  {format fout "  {define public {layout}:{return int int}~%"}
  {format fout "    {return self.width self.height}~%"}
  {format fout "    } | method: layout~%~%"}
  {format fout "  {define public {draw gc:GraphicContext}:void~%"}
  | OK, now call all the objects draw methods!
  {set group-objects {filter {lambda {x}:bool
                               | {return {not {isa CanvasGroupObject x}}}
                               {return true}}
                             canvas.Objects}}
  {map {lambda {x} {x.export-draw fout}} group-objects}

  {format fout "~%"}
  {format fout "    | Finally, a call to super.draw~%"}
  {format fout "    {super.draw gc}~%"}
  {format fout "    } | method: draw~%"}
  {format fout "  } | class class-number-~d~%~%" group-number}

  | and don't forget to close the file
  {fout.Close}
  } | export-menu-action


{define {save-menu-action canvas:NewCanvasSheet}:void
  filename:text
  urn:File
  fout:FileOutputPort
```

```
  | get the file ready.
  {set filename {popup-text-query "Please enter URN to save to: "}}
  {if {void? filename}
      {return}}
  {set urn {xresolve-filename filename}}
  {set fout {urn.write-open}}

  | do the copying
  {format fout "#CurlCanvas Version1.0~%"}
  {map {lambda {x}
         t:type={typeof x}
         {format fout "~%~p~%" {type->symbol t}}
         {x.save-to-file fout}
         }
       canvas.Objects}
  {format fout "EndObjects~%"}
  | and don't forget to close the file
  {fout.Close}
  } | save-menu-action


{define {load-menu-action canvas:NewCanvasSheet}:void
  filename:text
  urn:File
  fin:FileInputPort
  temp-text:text
  new-object-type:text
  objects:list
  new-object:NewCanvasObject

  | get the file ready.
  {set filename {popup-text-query "Please enter URN to load from: "}}
  {if {void? filename}
      {return}}
  {set urn {xresolve-filename filename}}
  {set fin {urn.read-open}}

  | do the loading
  {set temp-text {read-line fin}}
  {when {not {equal? temp-text "#CurlCanvas Version1.0"}}
    {fin.Close}
    {error "Bad format, unknown version number"}
    } | when

  {set new-object-type {read-string fin}}
  {while {not {equal? new-object-type "EndObjects"}}
```

```
    {if {void? new-object-type}
        {error "Bad format, unexpected end of file"}}
    {set new-object {eval {make-list 'alloc
                                     {text->symbol new-object-type}}}}
    {{cast {typeof new-object} new-object}.read-from-file fin}
    {set objects {cons new-object objects}}
    {set new-object-type {read-string fin}}
    } | while
  | and don't forget to close the file
  {fin.Close}

  | Ok, so now we have all these objects.  What now?
  {map {lambda {x} {x.add-to-canvas canvas}} objects}
  } | load-menu-action
```

A.2.3 constants.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| All sorts of constants.  Mostly, they fall under the categories of:
|      setting the position of a component
|      width/height of a component
|      toobar-button graphics
| There are also ruler-marker-tick separations and
| canvas-mouse-tolerance-distance.

{define-constant CANVAS-APP-WIDTH:int=502}
{define-constant CANVAS-APP-HEIGHT:int=475}

{define-constant CANVAS-SHEET-WIDTH:int=320}
{define-constant CANVAS-SHEET-HEIGHT:int=CANVAS-SHEET-WIDTH}

{define-constant RULER-HEIGHT:int=20}
{define-constant RULER-WIDTH:int=CANVAS-SHEET-WIDTH}

{define-constant
  CANVAS-SHEET-AND-RULERS-WIDTH:int={+ CANVAS-SHEET-WIDTH RULER-HEIGHT}}
{define-constant
  CANVAS-SHEET-AND-RULERS-HEIGHT:int={+ CANVAS-SHEET-HEIGHT RULER-HEIGHT}}
{define-constant CANVAS-SHEET-AND-RULERS-XPOS:int=100}
{define-constant CANVAS-SHEET-AND-RULERS-YPOS:int=65}

| distance between small tick marks, large marks, and their thickness
{define-constant RULER-TICKS-SMALL-SEP:int=10}
{define-constant RULER-TICKS-LARGE-SEP:int={* RULER-TICKS-SMALL-SEP 10}}
{define-constant RULER-TICKS-THICKNESS:int=1}
```

```
{define-constant RULER-MARKER-WIDTH:int=10}
{define-constant RULER-MARKER-HEIGHT:int=5}

{define-constant OBJECT-MARKER-WIDTH:int=5}
{define-constant OBJECT-MARKER-HEIGHT:int=OBJECT-MARKER-WIDTH}
{define-constant OBJECT-MARKER-COLOR:any='black}

{define-constant BOX-MARKER-WIDTH:int=10}
{define-constant BOX-MARKER-HEIGHT:int=BOX-MARKER-WIDTH}
{define-constant BOX-MARKER-COLOR:any='black}

{define-constant CANVAS-MOUSE-TOLERANCE-DISTANCE:float=5.0}



{define-constant MODE-BUTTON-WIDTH:int=30}
{define-constant MODE-BUTTON-HEIGHT:int=20}

{define-constant MODE-TOOLBAR-XPOS:int=-20}
{define-constant MODE-TOOLBAR-YPOS:int=100}

{define-constant PROP-BUTTON-WIDTH:int=40}
{define-constant PROP-BUTTON-HEIGHT:int=20}

{define-constant PROP-TOOLBAR-XPOS:int=200}
{define-constant PROP-TOOLBAR-YPOS:int=435}

{define-constant MENU-XPOS:int=0}
{define-constant MENU-YPOS:int=0}

{define-constant ERROR-BOX-WIDTH:int=229}
{define-constant ERROR-BOX-HEIGHT:int=25}
{define-constant ERROR-BOX-XPOS:int=CANVAS-SHEET-AND-RULERS-XPOS}
{define-constant ERROR-BOX-YPOS:int=40}
{define-constant DESCRIPTION-BOX-WIDTH:int=170}
{define-constant DESCRIPTION-BOX-HEIGHT:int=65}
{define-constant DESCRIPTION-BOX-XPOS:int=330}
{define-constant DESCRIPTION-BOX-YPOS:int=0}

{define-constant CMARK0:int=#x0001}  | vertex-marker
{define-constant CMARK1:int=#x0002}  | upper-left, 2
{define-constant CMARK2:int=#x0004}  | upper-middle, 4
{define-constant CMARK3:int=#x0008}  | upper-right, 8
{define-constant CMARK4:int=#x0010}  | middle-left, 16
{define-constant CMARK5:int=#x0020}  | center, 32
{define-constant CMARK6:int=#x0040}  | middle-right 64
```

```
{define-constant CMARK7:int=#x0080}  | lower-left 128
{define-constant CMARK8:int=#x0100}  | lower-middle 256
{define-constant CMARK9:int=#x0200}  | lower-right 512

{define-constant LAYERS-BELOW-CUR:int=#x0001}
{define-constant LAYERS-CUR:int=#x0002}
{define-constant LAYERS-ABOVE-CUR:int=#x0004}

{define-constant OVERLAP-PARTIAL:symbol='overlap}
{define-constant OVERLAP-NONE:symbol='non-intersecting}
{define-constant OVERLAP-2SURROUNDS1:symbol='2surrounds1}
{define-constant OVERLAP-1SURROUNDS2:symbol='1surrounds2}
{define-constant OVERLAP-CROSSSECTION:symbol='cross-section}
{define-constant OVERLAP-SAME:symbol='identical}

|| possible bits in CanvasObject.marker-type
{define-constant public 2POINTOBJECT:int=   #x0001}
{define-constant public RESIZEABLE:int=     #x0002}
{define-constant public FREEROTATEABLE:int= #x0004}
{define-constant public ROTATE90ONLY:int=   #x0008}
| the rest are currently unused: #x0040, #x0080, #x0100, #x0200,
| #x0400, #x0800, #x1000

{define-constant public HOLLOWBOX-MARKER-SHAPE:symbol='hollow-box}
{define-constant public FILLEDBOX-MARKER-SHAPE:symbol='filled-box}
{define-constant public HOLLOWCIRCLE-MARKER-SHAPE:symbol='hollow-circle}
{define-constant public FILLEDCIRCLE-MARKER-SHAPE:symbol='filled-circle}
```

A.2.4 edit.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This is all of the edit modes.

{define-class SelectEventHandler {NewCanvasEditMode}
  current-select-mode:symbol  | 'none, 'pressed, 'moving, 'select-box
  snap-start-p:Point
  start-p:Point
  last-p:Point
  last-selected-object:NewCanvasObject
  toggle-select:bool

  {define {init} {return}}

  {define {selected}:void
    {set self.current-select-mode 'none}
```

```
  } | method: selected

{define {set-selected-objects l:list}:void
  {map {lambda {x}
         {self.select-object x unselect-previous=false}
         } | lambda
       l}
  } | method: set-selected-objects

{define {cancel}:void
  {self.canvas.canvas.Sheet.add-select-box void}
  {return}
  } | method: cancel


{define {box-object obj:NewCanvasObject}:CanvasBox
  {return {new CanvasBox obj}}
  } | method: box-object


{define {select-object obj:NewCanvasObject
                       unselect-previous:bool=true}:CanvasBox
  cb:CanvasBox
  {set cb {self.box-object obj}}
  {self.canvas.canvas.Sheet.add-select-box
   cb unselect-previous=unselect-previous}
  {set self.last-selected-object obj}
  {return cb}
  } | method: select-object


{define public {pointer-press e:PointerEvent snap-point:Point}:void
  obj:NewCanvasObject
  shift-off:bool
  cb:CanvasBox
  already-selected:bool
  canvas:NewCanvasSheet=self.canvas.canvas.Sheet

  {super.pointer-press e snap-point}
  {if {not {eq? self.current-select-mode 'none}}
      {return}}
  {set shift-off {= 0 {bit-and e.state FShift}}}
  {set obj {canvas.closest-object {new Point e.x e.y}}}
  {when {void? obj}
    | no object near marker, select objects within a box
    {if shift-off
        | first unselect everything
```

```
      {canvas.add-select-box void}}
    | start the select box
    {set self.current-select-mode 'select-box}
    {set self.start-p {new Point e.x e.y}}
    {set self.last-p {new Point e.x e.y}}
    {set self.toggle-select {not shift-off}}
    {return}
    }
  | OK, if we're here, we are on an object
  | see if it was already selected
  {set cb {canvas.get-object-CanvasBox obj}}
  {if {void? cb}
      {set already-selected false}
      {set already-selected true}}
  {cond {{and {not already-selected} shift-off}
         | It wasn't already selected, so lets selected it
         {self.select-object obj unselect-previous=true}
         {set self.current-select-mode 'press-on-selected}
         }
        {{and {not already-selected} {not shift-off}}
         | It wasn't already selected, so lets add it
         {self.select-object obj unselect-previous=false}
         {set self.current-select-mode 'press-on-selected}
         }
        {{and already-selected {not shift-off}}
         | need to unselect it
         {cb.remove-from-canvas}
         {set self.current-select-mode 'press-on-unselected}
         }
        {else
         {set self.current-select-mode 'press-on-selected}
         }}
  | hey, don't forget to change member variables
  {set self.snap-start-p {snap-point.copy}}
  } | method: pointer-press


{define public {pointer-motion e:PointerEvent snap-point:Point}:void
  dist:int

  {super.pointer-motion e snap-point}
  {cond {{eq? self.current-select-mode 'none}
         {return}}
        {{eq? self.current-select-mode 'select-box}
         {rubberband-rectangle self.canvas.canvas.Sheet
                               self.start-p.x self.start-p.y
                               self.last-p.x self.last-p.y}
```

```
        {set self.last-p.x e.x}
        {set self.last-p.y e.y}
        {rubberband-rectangle self.canvas.canvas.Sheet
                              self.start-p.x self.start-p.y
                              self.last-p.x self.last-p.y}
        } | cond select-box
       } | cond
  } | method: pointer-motion


{define public {pointer-release e:PointerEvent snap-point:Point}:void
  upper-left:Point
  lower-right:Point
  contained-objects:list
  shift-off:bool
  full-enclose:bool
  canvas:NewCanvasSheet=self.canvas.canvas.Sheet

  {super.pointer-release e snap-point}
  | current-select-mode "can't" be 'none coming in.
  {when {eq? self.current-select-mode 'select-box}
    | before you forget, erase that rectangle
    {rubberband-rectangle canvas
                          self.start-p.x self.start-p.y
                          self.last-p.x self.last-p.y}
    | find the contained objects
    {set upper-left lower-right
         {twocorners->origdiag {new Point e.x e.y} self.start-p}}
    {set full-enclose
         {canvas.get-application-property 'select-by-enclosure}}
    {set contained-objects
         {filter {lambda {x}:bool
                   {cond {{not {void? {memq x canvas.hidden-objects}}}
                          {return false}}
                         {full-enclose
                          {return
                           {x.encloses-object upper-left lower-right}}}
                         {else
                          {return
                           {x.partially-encloses-object upper-left
                                                        lower-right}}}
                         }
                   } | lambda
                 canvas.Objects}}

    | now for the actual selection
    {cond {self.toggle-select
```

```
          | toggle everything
          {map {lambda {x}
                 box:CanvasBox
                 {set box {canvas.get-object-CanvasBox x}}
                 {if {void? box} | if true, not selected
                     {self.select-object x unselect-previous=false}
                     {box.remove-from-canvas}}
                 } | lambda
               contained-objects}
          } | cond toggle-select
         {else
          | remove everything previously selected
          {canvas.add-select-box void}
          | now start adding
          {map {lambda {x}
                 {self.select-object x unselect-previous=false}
                 } | lambda
               contained-objects}
          } | cond else
         } | cond
    } | when

  {if {or {eq? self.current-select-mode 'select-box}
          {eq? self.current-select-mode 'press-on-selected}
          {eq? self.current-select-mode 'press-on-unselected}}
      {set self.current-select-mode 'none}}
  } | method: pointer-release

{define {esc-key-press}:void
  | Only want to deal if we're not in the middle of something
  {if {eq? self.current-select-mode 'none}
      {super.esc-key-press}}
  } | method: esc-key-press

{define public {key-press e:KeyEvent}:void
  | if we're busy, don't erase things
  | I didn't use the state of the KeyEvent just in case we decide on
  | a two click event
  canvas:NewCanvasSheet=self.canvas.canvas.Sheet
  {super.key-press e}
  {if {not {eq? self.current-select-mode 'none}}
      {return}}
  | only backspace does something -- which is to delete all objects
  {when {or {= e.key 127} {= e.key 8}}  | backspace/delete
    {map {lambda {x} | where x is a CanvasBox
           {if x.good-object-for-box
               {x.obj.remove-from-canvas}
```

```
                }
            }
          canvas.select-boxes}
      {canvas.commit-exchange}
      } | when
    } | method: key-press

  } | class: SelectEventHandler


{define-class SelectResizeEventHandler {SelectEventHandler}
  {define {box-object obj:NewCanvasObject}:CanvasBox
    {return {new ResizeBox obj}}
    }

  {define {selected}:void
    {super.selected}
    {self.canvas.canvas.Sheet.message "Resize mode selected"}
    {self.canvas.canvas.Sheet.set-event-descriptions
     pointer-press="Resize objects"
     shiftify="Square Resize"}
    }
  }

{define-class SelectRotateEventHandler {SelectEventHandler}
  {define {box-object obj:NewCanvasObject}:CanvasBox
    {return {new RotateBox obj}}
    }

  {define {selected}:void
    {super.selected}
    {self.canvas.canvas.Sheet.message "Rotate mode selected"}
    {self.canvas.canvas.Sheet.set-event-descriptions
     pointer-press="Rotate objects"
     shiftify=""}
    }
  }

{define-class SelectMoveEventHandler {SelectEventHandler}
  {define {box-object obj:NewCanvasObject}:CanvasBox
    {return {new CanvasMoveBox obj}}
    }

  {define {selected}:void
    {super.selected}
    {self.canvas.canvas.Sheet.message "Move mode selected"}
    {self.canvas.canvas.Sheet.set-event-descriptions
```

```
    pointer-press="Move objects"
    shiftify="Horiz/Vert Only"}
   }
 }
```

A.2.5 geom.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains all the random geometry stuff.  Distance between
| a point and a line, conversion between radians and degrees,
| rotation, overlap, etc.

{define {two-point-distance p1:Point p2:Point}:float
  {return {sqrt {+ {square {- p1.x p2.x}} {square {- p1.y p2.y}}}}}
  } | procedure: two-point-distance

{define {degree->radian ang:float}:float
  {return {/ {* ang PI} 180}}}
{define {radian->degree ang:float}:float
  {return {/ {* 180 ang} PI}}}


{define {twocorners->origdiag p1:Point p2:Point
                              pixel-orientation:bool=true
                              }:{return Point Point}
  | takes two corners of a rectangle and returns the rectangle's upper
  | left and lower right
  | bugbug: ignores problems with floats as points, since min and max
  | deal int-ly
  minx:int
  miny:int
  maxx:int
  maxy:int

  {set minx p1.x}
  {set maxx p1.x}
  {set miny p1.y}
  {set maxy p1.y}
  {set minx {min minx p2.x}}
  {set maxx {max maxx p2.x}}
  {set miny {min miny p2.y}}
  {set maxy {max maxy p2.y}}
  {if pixel-orientation
      {return {new Point minx miny} {new Point maxx maxy}}
      {return {new Point minx maxy} {new Point maxx miny}}
```

```
    }
  } | proc: twocorners->origdiag


{define {point-to-lineseg-distance p:Point p1:Point p2:Point}:float
  | Shortest distance between a point and a line segment (specified
  | by the two end-points).

  | general:
  | if you have line-seg(AB) and point(x)
  |  make lineseg(xp) such that point(P) is on line(AB) and
  |            lineseg(xp) is perpendicular to line(AB)
  | if point(p) is on lineseg(ab), return len(lineseg(xp))
  | else return (min (leng(lineseg(xa)), len(lineseg(xb)))

  | from xfig source (u_geom):
  |   if (xp, yp) is your point, and your line segment is between
  |   (x1, y1) and (x2, y2), then the shortest distance between the
  |   two is through the point
  |      y = (s*(xp - x1 + s * y1) + yp) / (1 + s * s)
  |      x = x1 + s * (y - y1)
  |   where s is the inverse-slope.  (dx / dy)
  | This only works when there is slope.  Otherwise, do it the
  | simple way.
  | Either way, you need to check to make sure that your point is on
  | the line, otherwise take the min of the distances between the
  | point and the two endpoints of the line segment.

  inv-slope:float    | inverse slope
  intersection:Point | the point where the shortest line to (p1, p2)
                     | from p would intersect (p1, p2)
  on-segment:bool    | if intersection is on the segment from p1 to p2

  {cond {{= p1.y p2.y}                        | horizontal line
         {set intersection {new Point p.x p1.y}}
         } | horizontal line
        {{= p1.x p2.x}                        | vertical line
         | note, this isn't necessary, just a bit faster
         {set intersection {new Point p1.x p.y}}
         }
        {else
         {let inter-y:float
              inter-x:float

           {set inv-slope {/ {cast float {- p1.x p2.x}} {- p1.y p2.y}}}
           {set inter-y
                {/ {+ {* inv-slope
```

```
                    {+ p.x
                       {- p1.x}
                       {* inv-slope p1.y}}}
                 p.y}
              {+ 1
                 {square inv-slope}}}}
        {set inter-x
             {+ p1.x
                {* inv-slope
                   {- inter-y p1.y}}}}
        {set intersection {new Point {round inter-x} {round inter-y}}}
        } |let
       } | else
      } | cond

| now you have the intersection point for the perpendicular line.
| See if that point is part of the line segment from p1 to p2.
{cond {{= 0 inv-slope}
       | this is the initial value.  Therefore, horizontal or vertical
       {if {and | makes it both x-wise and y-wise
            {or {and {>= p1.x intersection.x} {<= p2.x intersection.x}}
                {and {>= p2.x intersection.x} {<= p1.x intersection.x}}}
            {or {and {>= p1.y intersection.y} {<= p2.y intersection.y}}
                {and {>= p2.y intersection.y} {<= p1.y intersection.y}}}
            } | and
           {set on-segment true}
           {set on-segment false}
           } | if
       } | cond = 0
      {{< 0 inv-slope}
       | this is negative slope.
       {if {or
            | first x2 is in the upper-left
            {and {>= p2.x intersection.x} {>= p2.y intersection.y}
                 {<= p1.x intersection.x} {<= p1.y intersection.y}}
            | now x1 is in the upper-left
            {and {>= p1.x intersection.x} {>= p1.y intersection.y}
                 {<= p2.x intersection.x} {<= p2.y intersection.y}}
            } | or
           {set on-segment true}
           {set on-segment false}
           } | if
       } | cond negative slope
      {else
       | positive slope
       {if {or
            | first x2 is in the lower-left
```

```
            {and {<= p2.x intersection.x} {>= p2.y intersection.y}
                 {>= p1.x intersection.x} {<= p1.y intersection.y}}
            | now x1 is in the lower-left
            {and {<= p1.x intersection.x} {>= p1.y intersection.y}
                 {>= p2.x intersection.x} {<= p2.y intersection.y}}
            } | or
           {set on-segment true}
           {set on-segment false}
           } | if
       } | else
      } | cond

  | if it's on the line segment, return the distance
  {if on-segment
      {return {two-point-distance p intersection}}
      }

  | no, then one of the endpoints is the closest point
  {return {min {two-point-distance p p1}
               {two-point-distance p p2}}}

  } | method: point-to-lineseg-distance


{define {twopt->linefunction p1:Point p2:Point}:{proc {float}:float}
  dy:int
  dx:int
  slope:float
  own-copy-of-p1:Point
  ans:{proc {float}:float}

  {set dy {- p2.y p1.y}}
  {set dx {- p2.x p1.x}}
  {cond {{= 0 dx}
         | vertical line
         {throw {new Exception "No function for line.  Vertical line."}}
         }
        {{= 0 dy}
         | horizontal
         {let y:int=p2.y
           {return {lambda {x:float}:float {return y}}}}
         }
        {else
         {set slope {/ {cast float dy} dx}}
         {set own-copy-of-p1 {p1.copy}}
         {set ans
              {lambda {x:float}:float
```

```
              {return {round {+ {* slope
                                   {- x own-copy-of-p1.x}}
                                own-copy-of-p1.y}}}}
        {return ans}
        }} | else-cond
  } | twopt->linefunction


{define {two-linesegs-cross? obj1-start:Point obj1-end:Point
                             obj2-start:Point obj2-end:Point
                             equal-counts:bool=true}:bool
  | Just sees if obj2-start and obj2-end are on opposite sides of
  | the line defined by obj1-start and obj1-end.  To do this,
  | creates a function that describes the line.  Then computes the y
  | coord of that line at both obj2-start and obj2-end.  Using the
  | notation o1(foo) for "The y coord of obj1 at x=foo", sx2 for
  | "obj2-start.x, and "ex2 for "obj2-end.x", we return
  |     (0 < ((o1(sx2) - sy2) * (o1(ex2) - ey2)))
  obj1-func:{proc {float}:float}
  obj2-func:{proc {float}:float}
  result:float

  {try
   {set obj1-func {twopt->linefunction obj1-start obj1-end}}
   {catch {e:Exception}
          {try
           {set obj2-func {twopt->linefunction obj2-start obj2-end}}
           {catch {e:Exception}}
           } | try
          } | catch
   } | try

  {cond {{not {void? obj1-func}}
         {set result {* {- {obj1-func obj2-start.x} obj2-start.y}
                        {- {obj1-func obj2-end.x} obj2-end.y}}}
         }
        {{not {void? obj2-func}}
         {set result {>= 0 {* {- {obj2-func obj1-start.x} obj1-start.y}
                              {- {obj2-func obj1-end.x} obj1-end.y}}}}
         }
        {else
         | two vertical lines
         {let o1min:int={min obj1-start.x obj1-end.x}
              o1max:int={max obj1-start.x obj1-end.x}
              o2min:int={min obj2-start.x obj2-end.x}
              o2max:int={max obj2-start.x obj2-end.x}
              overlap:symbol={find-1D-overlap o1min o1max
```

```
                                                    o2min o2max
                                                    equal-counts=equal-counts}
          {if {eq? OVERLAP-NONE overlap}
              {return false}
              {return true}
              }
             } | let
         } | else
        } | cond

  {if equal-counts
      {return {>= 0 result}}
      {return {>  0 result}}
      }
  } | method: two-linesegs-cross?


{define {find-1D-overlap obj1-min:int obj1-max:int
                         obj2-min:int obj2-max:int
                         equal-counts:bool=true
                         equal-completes:bool=true}:symbol
  ans:symbol

  {if {and {not equal-counts}
           {or {= obj2-max obj1-min}
               {= obj1-max obj2-min}}}
      {return OVERLAP-NONE}}

  {cond {{and {= obj2-min obj1-min} {= obj2-max obj1-max}}
         {set ans OVERLAP-SAME}}
        {{< obj2-min obj1-min}
         | obj2 starts to the left/top of obj1
         {cond {{< obj2-max obj1-min}
                | all to the left of the object
                {set ans OVERLAP-NONE}
                }
               {{<= obj2-max obj1-max}
                | overlap, at least for now
                {set ans OVERLAP-PARTIAL}
                }
                | surrounds the x-coords of the object
               {else
                {set ans OVERLAP-2SURROUNDS1}
                }
               } | inner-cond
         } | cond-starts-left
        {{< obj2-min obj1-max}
```

```
        | obj2 starts in the middle of object1, or on it
        {cond {{= obj2-min obj1-min}
               | set it to partial for now
               {set ans OVERLAP-PARTIAL}
               }
              {{< obj2-max obj1-max}
               | rect is inside object
               {set ans OVERLAP-1SURROUNDS2}
               }
              {else
               | it's gotta overlap
               {set ans OVERLAP-PARTIAL}
               }
              } | inner-cond
        } | cond-middle
       {{= obj2-min obj1-max}
        | set it to partial for now
        {set ans OVERLAP-PARTIAL}
        }
       {else
        | rect is just to the left of object
        {set ans OVERLAP-NONE}
        }
       } | outer cond
  {if {and {eq? ans OVERLAP-PARTIAL} equal-completes}
      {cond {{= obj2-min obj1-min}
             {if {< obj2-max obj1-max}
                 {set ans OVERLAP-1SURROUNDS2}
                 {set ans OVERLAP-2SURROUNDS1}}
             }
            {{= obj2-max obj1-max}
             {if {< obj2-min obj1-min}
                 {set ans OVERLAP-2SURROUNDS1}
                 {set ans OVERLAP-1SURROUNDS2}}
             }
            }
      }
  {return ans}
  } | method: find-1D-overlap


{define {find-2D-rect-overlap obj1-corner:Point obj1-diag:Point
                              obj2-corner:Point obj2-diag:Point
                              equal-counts:bool=true
                              equal-completes:bool=true}:symbol
  x-overlap:symbol
  y-overlap:symbol
```

```
ans:symbol

{set obj1-corner obj1-diag {twocorners->origdiag obj1-corner obj1-diag}}
{set obj2-corner obj2-diag {twocorners->origdiag obj2-corner obj2-diag}}
{set x-overlap {find-1D-overlap obj1-corner.x obj1-diag.x
                                obj2-corner.x obj2-diag.x
                                equal-counts=equal-counts
                                equal-completes=equal-completes}}
{set y-overlap {find-1D-overlap obj1-corner.y obj1-diag.y
                                obj2-corner.y obj2-diag.y
                                equal-counts=equal-counts
                                equal-completes=equal-completes}}

| we have OVERLAP-PARTIAL, OVERLAP-2SURROUNDS1, and OVERLAP-1SURROUNDS2.
{cond {{or {eq? x-overlap OVERLAP-NONE}
           {eq? y-overlap OVERLAP-NONE}}
       {set ans OVERLAP-NONE}}
      {{or {eq? x-overlap OVERLAP-PARTIAL}
           {eq? y-overlap OVERLAP-PARTIAL}}
       | since neither is OVERLAP-NONE, crosses at least one of the
       | sides, possibly two adjacent sides
       {set ans OVERLAP-PARTIAL}}
      {{or {and {eq? x-overlap OVERLAP-2SURROUNDS1}
                {eq? y-overlap OVERLAP-1SURROUNDS2}}
           {and {eq? x-overlap OVERLAP-1SURROUNDS2}
                {eq? y-overlap OVERLAP-2SURROUNDS1}}}
       | a "cross-section"
       {set ans OVERLAP-CROSSSECTION}}
      {{and {eq? x-overlap OVERLAP-2SURROUNDS1}
            {eq? y-overlap OVERLAP-2SURROUNDS1}}
       | encloses
       {set ans OVERLAP-2SURROUNDS1}}
      {{or {eq? x-overlap OVERLAP-SAME} {eq? y-overlap OVERLAP-SAME}}
       {cond {{and {eq? x-overlap OVERLAP-SAME}
                   {eq? y-overlap OVERLAP-SAME}}
              {set ans OVERLAP-SAME}
              }
             {{or {eq? x-overlap OVERLAP-2SURROUNDS1}
                  {eq? y-overlap OVERLAP-2SURROUNDS1}}
              {set ans OVERLAP-2SURROUNDS1}
              }
             {else | the other is OVERLAP-1SURROUNDS2
              {set ans OVERLAP-1SURROUNDS2}
              }} | else-cond
       }
      {else | both are "OVERLAP-1SURROUNDS2.
       {set ans OVERLAP-1SURROUNDS2}}
```

```
      } | cond
  {if {and {eq? ans OVERLAP-PARTIAL}
           {not {and {eq? x-overlap OVERLAP-PARTIAL}
                     {eq? y-overlap OVERLAP-PARTIAL}}}}
      {set ans OVERLAP-CROSSSECTION}
      }

  {return ans}
  } | method: find-2D-rect-overlap


{define {find-2D-linerect-overlap line-start:Point line-end:Point
                                  corner:Point diagcorner:Point}:symbol
  temp-num:float
  rect-overlap:symbol

  {set corner diagcorner {twocorners->origdiag corner diagcorner}}
  {set rect-overlap
       {find-2D-rect-overlap line-start line-end corner diagcorner}}
  {cond {{eq? rect-overlap OVERLAP-NONE}
         {return OVERLAP-NONE}}
        {{eq? rect-overlap OVERLAP-2SURROUNDS1}
         {return OVERLAP-2SURROUNDS1}}
        {else
         | OK, if we're here, we have to do some real work to find out
         | the answer.  What I'm going to do is see if the line segment
         | between corner1 and corner3 or the line segment between
         | corner2 and corner4 intersect the line segment between p1 and
         {if {two-linesegs-cross? line-start line-end corner diagcorner}
             {return OVERLAP-PARTIAL}}

         | we already had the main diag, switch
         {set temp-num corner.x}
         {set corner.x diagcorner.x}
         {set diagcorner.x temp-num}
         {if {two-linesegs-cross? line-start line-end corner diagcorner}
             {return OVERLAP-PARTIAL}}
         } | else
        } | cond
  {return OVERLAP-NONE}
  } | procedure: find-2D-linerect-overlap


{define {rotate-point p:Point
                      deg-angle:float=0 rad-angle:float=0
                      center:Point={new Point 0 0}
                      pixel-orientation:bool=true}:Point
```

```
  dx:float
  dy:float
  angle:float
  mag:float

  {if {!= deg-angle 0}
      {set rad-angle {degree->radian deg-angle}}}
  {if {= rad-angle 0}
      {return {new Point p.x p.y}}}
  {set dx {- p.x center.x}}
  {set dy {- p.y center.y}}
  {if {and {= dx 0} {= dy 0}}
      {return {new Point p.x p.y}}}
  {if pixel-orientation
      {set angle {arctangent2 {- dy} dx}}
      {set angle {arctangent2 dy dx}}
      }
  {set angle {+ angle rad-angle}}
  {set mag {two-point-distance p center}}
  {set dx {* mag {cosine angle}}}
  {set dy {* mag {sine angle}}}
  {when p.int-numbers
    {set dx {round dx}}
    {set dy {round dy}}
    } | when
  {if pixel-orientation
      {set dy {- dy}}
      }
  {return {new Point
               {+ center.x dx} {+ center.y dy}
               int-numbers=p.int-numbers}}
  } | procedure: rotate-point


{define {centerandcorner->diag center:Point corner:Point}:Point
  x:int
  y:int

  {set x {- center.x {- corner.x center.x}}}
  {set y {- center.y {- corner.y center.y}}}
  {return {new Point x y}}
  } | procedure: centerandcorner->diag


{define public {points-minmax pts:list}:{return int int int int}
  xmin:int
  xmax:int
```

```
  ymin:int
  ymax:int
  templist:list

  | set up "bounding box".  Start with first values.
  {set xmin {car pts}.x}
  {set xmax xmin}
  {set ymin {car pts}.y}
  {set ymax ymin}
  {set templist pts}
  | now, go through the rest of the list and set the mins and maxes
  {map
   {lambda {p}                          | where p is a point
     {if {< p.x xmin}
         {set xmin p.x}
         {if {> p.x xmax}
             {set xmax p.x}
             }
         }
     {if {< p.y ymin}
         {set ymin p.y}
         {if {> p.y ymax}
             {set ymax p.y}
             }
         }
     } | lambda
   templist
   }
  {return xmin xmax ymin ymax}
  } | points-minmax
```

A.3 Objects Found in the Drawing Sheet

A.3.1 cbox.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| Defined in here are all of the different types of selection boxes.

{define-class CanvasBox {NewCanvasObject}
  obj:NewCanvasObject
  corners-array:{array Point}
  rotated:bool

  extra-width:int
```

```
extra-height:int
show-border:bool
good-object-for-box:bool

{define {init obj:NewCanvasObject
              extra-width:int=0
              extra-height:int=0
              show-border:bool=false}
  {set self.corners-array {new {array Point} 4}}
  {set self.extra-width extra-width}
  {set self.extra-height extra-height}
  {set self.show-border show-border}
  {set self.obj obj}
  {set self.good-object-for-box true}
  {self.set-canvas obj.canvas} | which sets self.rotated
  {self.internal-initialize} | which calls super.init
  } | method: init


{define {change-object obj}:void
  {set self.obj obj}
  {self.re-initialize}
  } | method: change-object

{define private {internal-initialize}:void
  width:int
  height:int
  half-extra-width:int={/ self.extra-width 2}
  half-extra-height:int={/ self.extra-height 2}
  hack0:Point
  hack1:Point
  hack2:Point
  hack3:Point
  xmax:int
  xmin:int
  ymax:int
  ymin:int

  | do NOT append the object to yourself.  Leave it in the canvas.
  | You just frame the object.

  | hack: can't use aref directly when doing this set...
  {if self.rotated
      {set hack0 hack1 hack2 hack3
           {self.obj.four-corners rotated=true}}
      {set hack0 hack1 hack2 hack3
           {self.obj.four-corners rotated=false}}
```

```
    }
  {set {aref self.corners-array 0} hack0}
  {set {aref self.corners-array 1} hack1}
  {set {aref self.corners-array 2} hack2}
  {set {aref self.corners-array 3} hack3}

  | figure out the width and height of yourself
  {if {!= 0 {bit-and self.obj.marker-type 2POINTOBJECT}}
      {set xmin xmax ymin ymax {points-minmax
                                {make-list hack0 hack1}}}
      {set xmin xmax ymin ymax {points-minmax
                                {make-list hack0 hack1 hack2 hack3}}}
      }
  {set width {+ {- xmax xmin} self.extra-width}}
  {set height {+ {- ymax ymin} self.extra-height}}
  {super.init {- xmin half-extra-width}
              {- ymin half-extra-height}
              width height}
  | hack: set the depth to be 0 manually, since most objects
  | shouldn't be allowed.
  {super.change-object-property 'depth 0}


  {self.invalidate}
  } | method: internal-initialize

{define {re-initialize}:void
  {invoke-method CanvasBox 'internal-initialize self}
  } | method: re-initialize

{define {show}:void
  {set self.show-border true}
  {self.invalidate}
  } | method: show
{define {hide}:void
  {set self.show-border false}
  {self.invalidate}
  } | method: hide

{define {add-to-canvas canvas:NewCanvasSheet copy-properties:bool=false
                       commit:bool=false}:void
  | We can't get passed "unselect-previous", so just don't use this guy
  | Put the burden on Canvas.
  {error "Add CanvasBox directly using CanvasSheet.add-select-box"}
  } | method: add-to-canvas

{define {set-canvas canvas:NewCanvasSheet}:void
```

```
  {set self.canvas canvas}
  {set self.rotated {canvas.get-application-property
                     'rotate-selection-boxes}}
  } | method: set-canvas

{define {remove-from-canvas}:void
  {self.canvas.remove-select-box self}
  } | method: remove-from-canvas

{define public {draw gc:GraphicContext}:void
  p1:Point
  p2:Point
  p3:Point
  p4:Point
  {when {and self.show-border {= 0 {bit-and self.obj.marker-type
                                            2POINTOBJECT}}}
    | It has 4 corners.  Just draw between them.
    {set p1
         {self.canvas-point-to-local-coord {aref self.corners-array 0}}}
    {set p2
         {self.canvas-point-to-local-coord {aref self.corners-array 1}}}
    {set p3
         {self.canvas-point-to-local-coord {aref self.corners-array 2}}}
    {set p4
         {self.canvas-point-to-local-coord {aref self.corners-array 3}}}
    {gc.draw-line p1.x p1.y p2.x p2.y 'black 1}
    {gc.draw-line p2.x p2.y p3.x p3.y 'black 1}
    {gc.draw-line p3.x p3.y p4.x p4.y 'black 1}
    {gc.draw-line p4.x p4.y p1.x p1.y 'black 1}
    } | when
  } | method: draw

{define {copy}:NewCanvasObject
  {return {self.obj.copy}}
  } | method: copy

{define {change-object-property prop:symbol newval:any}:void
  {self.obj.change-object-property prop newval}
  } | method: change-canvas-property

{define {init-marker m:NewCanvasSelectMarker}:void
  | the following sets the canvas, if it's not void.  Also, this causes
  | the marker to append itself to the canvas
  {m.set-box self}
  } | method: init-marker

{define {set-canvas-for-marker m:NewCanvasSelectMarker}:void
```

```
    | the following causes the marker to append itself to the canvas, too
    {m.set-canvas self.canvas}
    } | method: set-canvas-for-marker

  {define {append-marker m:NewCanvasSelectMarker}:void
    {m.show}
    } | method: append-marker

  {define {remove-marker m:NewCanvasSelectMarker}:void
    {m.hide}
    } | method: remove-marker
  } | class: CanvasBox



{define-class CanvasMoveBox {CanvasBox}
  move-marker:MoveBoxMarker

  {define {init show-move-marker:bool=false ...}
    {myapply-method CanvasBox 'init self {rest-as-list}}
    {set self.move-marker {new MoveBoxMarker
                               x-center=self.object-center.x
                               y-center=self.object-center.y}}
    {if {eq? {typeof self} CanvasMoveBox}
        {super.show}
        }

    {self.init-marker self.move-marker}
    {self.move-marker.show}
    } | method: init


  {define {re-initialize}:void
    {super.re-initialize}
    {self.move-marker.set-position x=self.object-center.x
                                   y=self.object-center.y}
    } | re-initialize

  {define {set-canvas canvas:NewCanvasSheet}:void
    {super.set-canvas canvas}
    {if {not {void? self.move-marker}}
        {self.set-canvas-for-marker self.move-marker}}
    } | method: set-canvas

  {define {remove-from-canvas}:void
    {super.remove-from-canvas}
    {self.canvas.remove-object self.move-marker}
```

```
    } | remove-from-canvas
  } | class: CanvasMoveBox


{define-mixin-class CanvasBoxWithMarkers {CanvasBox}
  markers:DynamicStructure

  {define {init ...}
    {set self.markers {new DynamicStructure}}
    {myapply-method CanvasBox 'init self {rest-as-list}}
    } | method: init


  {define {add-marker-array name:symbol the-array:any}:void
    {self.markers.add-name name value=the-array}
    } | method: add-marker-array


  {define {add-marker array-name:symbol marker-type:type
                      marker-pos:int array-pos:int}:void
    marker:NewCanvasSelectMarker
    temp-array:any

    {set marker {new marker-type 0 0 marker-pos}}
    {marker.set-box self}
    {set temp-array {self.markers.get-value array-name}}
    {set {aref temp-array array-pos} marker}
    | note, you don't need to "set-value", since you mutated the array
    {self.init-marker marker}
    } | method: add-marker

  {define {re-initialize}:void
    {super.re-initialize}
    {self.change-markers-obj}
    } | method: re-initialize

  {define {hide-all-markers}:void
    names-list:list
    markers-array:any
    {set names-list {self.markers.list-names}}
    {map {lambda {x}   | where x is a name
           {set markers-array {self.markers.get-value x}}
           {dotimes {i {length markers-array}}
             {if {not {void? {aref markers-array i}}}
                 {{aref markers-array i}.hide}
                 } | if
             } | dotimes
```

```
        } | lambda
      names-list}
  } | hide-all-markers

{define {show-all-markers}:void
  names-list:list
  markers-array:any
  {set names-list {self.markers.list-names}}
  {map {lambda {x}   | where x is a name
         {set markers-array {self.markers.get-value x}}
         {dotimes {i {length markers-array}}
           {if {not {void? {aref markers-array i}}}
               {{aref markers-array i}.show}
               } | if
           } | dotimes
         } | lambda
       names-list}
  } | show-all-markers

{define {change-markers-obj}:void
  names-list:list
  markers-array:any
  {set names-list {self.markers.list-names}}
  {map {lambda {x}   | where x is a name
         {set markers-array {self.markers.get-value x}}
         {dotimes {i {length markers-array}}
           {if {not {void? {aref markers-array i}}}
               {set {aref markers-array i}.obj self.obj}
               } | if
           } | dotimes
         } | lambda
       names-list}
  } | method: change-markers-obj

{define {set-canvas canvas:NewCanvasSheet}:void
  names-list:list
  markers-array:any
  {super.set-canvas canvas}
  {set names-list {self.markers.list-names}}
  {map {lambda {x}   | where x is a name
         {set markers-array {self.markers.get-value x}}
         {dotimes {i {length markers-array}}
           {if {not {void? {aref markers-array i}}}
               {self.set-canvas-for-marker {aref markers-array i}}
               } | if
           } | dotimes
         } | lambda
```

```
        names-list}
    } | method: set-canvas

  {define {remove-from-canvas}:void
    names-list:list
    markers-array:any
    {super.remove-from-canvas}
    {set names-list {self.markers.list-names}}
    {map {lambda {x}   | where x is a name
           {set markers-array {self.markers.get-value x}}
           {dotimes {i {length markers-array}}
             {if {not {void? {aref markers-array i}}}
                 {{aref markers-array i}.hide}
                 } | if
             } | dotimes
           } | lambda
         names-list}
    } | remove-from-canvas
  } | class: CanvasBoxWithMarkers



{define-class Canvas4Box {CanvasBoxWithMarkers}

  {define {init obj:NewCanvasObject marker-type:type ...}
    {generic-apply-method CanvasBoxWithMarkers 'init
                          self
                          {rest-as-list}
                          obj
                          extra-width=BOX-MARKER-WIDTH
                          extra-height=BOX-MARKER-HEIGHT}
    | now the markers
    | hack: can't make an array of "marker-type", so make it an any
    {self.add-marker-array 'corner-markers {new {array any} 4}}
    {self.add-marker 'corner-markers marker-type CMARK1 0}
    {self.add-marker 'corner-markers marker-type CMARK3 1}
    {self.add-marker 'corner-markers marker-type CMARK7 2}
    {self.add-marker 'corner-markers marker-type CMARK9 3}
    {invoke-method Canvas4Box 'move-markers self}
    } | method: init

  {define {re-initialize}:void
    {super.re-initialize}
    {invoke-method Canvas4Box 'move-markers self}
    } | method: re-initialize

  {define {get-marker-of-pos pos:int}:NewCanvasSelectMarker
```

```
  marker-array:any

  {set marker-array {self.markers.get-value 'corner-markers}}
  {cond {{= pos CMARK1} {return {aref marker-array 0}}}
        {{= pos CMARK3} {return {aref marker-array 1}}}
        {{= pos CMARK7} {return {aref marker-array 2}}}
        {{= pos CMARK9} {return {aref marker-array 3}}}
        {else {return void}}
        } | cond
  } | method: get-marker-of-pos


{define private {move-markers}:void
  p1:Point
  p2:Point
  p3:Point
  p4:Point
  mark1:NewCanvasSelectMarker
  mark3:NewCanvasSelectMarker
  mark7:NewCanvasSelectMarker
  mark9:NewCanvasSelectMarker

  {set mark1 {invoke-method Canvas4Box 'get-marker-of-pos self CMARK1}}
  {set mark3 {invoke-method Canvas4Box 'get-marker-of-pos self CMARK3}}
  {set mark7 {invoke-method Canvas4Box 'get-marker-of-pos self CMARK7}}
  {set mark9 {invoke-method Canvas4Box 'get-marker-of-pos self CMARK9}}

  {cond {{!= 0 {bit-and self.obj.marker-type 2POINTOBJECT}}
         | It has only 2 points, not 4
         {set p1 {aref self.corners-array 0}}
         {set p2 {aref self.corners-array 1}}
         | do we have an upper-left and lower-right?  Or the other diag?
         {cond {{< p1.x p2.x}
                | it's the upper-left/lower-right diag
                {mark1.set-position x=p1.x y=p1.y}
                {mark1.show}
                {mark9.set-position x=p2.x y=p2.y}
                {mark9.show}
                {mark3.hide}
                {mark7.hide}
                }
               {else
                {mark3.set-position x=p1.x y=p1.y}
                {mark3.show}
                {mark7.set-position x=p2.x y=p2.y}
                {mark7.show}
                {mark1.hide}
```

```
                {mark9.hide}
                }
               } | cond
          }
         {else
          {set p1 {aref self.corners-array 0}}
          {set p2 {aref self.corners-array 1}}
          {set p3 {aref self.corners-array 2}}
          {set p4 {aref self.corners-array 3}}
          {mark1.set-position x=p1.x y=p1.y}
          {mark3.set-position x=p2.x y=p2.y}
          {mark7.set-position x=p4.x y=p4.y}
          {mark9.set-position x=p3.x y=p3.y}
          | and just to be safe, make sure that they're showing
          {mark1.show}
          {mark3.show}
          {mark7.show}
          {mark9.show}
          } | else
         } | cond
    } | method: move-markers
  } | class: Canvas4Box



{define-class Canvas8Box {Canvas4Box}
  {define {init obj:NewCanvasObject marker-type:type ...}
    {generic-apply-method Canvas4Box 'init self
                          {rest-as-list}
                          obj marker-type
                          extra-width=BOX-MARKER-WIDTH
                          extra-height=BOX-MARKER-HEIGHT}
    | do the markers
    | hack: can't make an array of "marker-type", so make it an any
    {self.add-marker-array 'midpoint-markers {new {array any} 4}}
    {self.add-marker 'midpoint-markers marker-type CMARK2 0}
    {self.add-marker 'midpoint-markers marker-type CMARK4 1}
    {self.add-marker 'midpoint-markers marker-type CMARK6 2}
    {self.add-marker 'midpoint-markers marker-type CMARK8 3}
    {invoke-method Canvas8Box 'move-markers self}
    } | method: init

  {define {get-marker-of-pos pos:int}:NewCanvasSelectMarker
    marker-array:any

    {set marker-array {self.markers.get-value 'midpoint-markers}}
    {cond {{= pos CMARK2} {return {aref marker-array 0}}}
```

```
        {{= pos CMARK4} {return {aref marker-array 1}}}
        {{= pos CMARK6} {return {aref marker-array 2}}}
        {{= pos CMARK8} {return {aref marker-array 3}}}
        {else {return {super.get-marker-of-pos pos}}}
        } | cond
  } | method: get-marker-of-pos


{define {re-initialize}:void
  {super.re-initialize}
  {invoke-method Canvas8Box 'move-markers self}
  } | method: re-initialize


{define private {move-markers}:void
  p1:Point
  p2:Point
  p3:Point
  p4:Point
  p1p2:Point
  p2p3:Point
  p3p4:Point
  p4p1:Point

  mark2:NewCanvasSelectMarker
  mark4:NewCanvasSelectMarker
  mark6:NewCanvasSelectMarker
  mark8:NewCanvasSelectMarker

  mark1:NewCanvasSelectMarker
  mark3:NewCanvasSelectMarker
  mark7:NewCanvasSelectMarker
  mark9:NewCanvasSelectMarker

  hedge-width:int
  hedge-height:int
  vedge-width:int
  vedge-height:int


  {set mark2 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK2}}
  {set mark4 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK4}}
  {set mark6 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK6}}
  {set mark8 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK8}}

  {set mark1 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK1}}
  {set mark3 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK3}}
```

```
{set mark7 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK7}}
{set mark9 {invoke-method Canvas8Box 'get-marker-of-pos self CMARK9}}


{cond {{!= 0 {bit-and self.obj.marker-type 2POINTOBJECT}}
       | It has only 2 points, not 4.  So you better hide your guys.
       {mark2.hide}
       {mark4.hide}
       {mark6.hide}
       {mark8.hide}
       | and now those corner guys have to take on extra roles
       {set mark1.marker-position {bit-or CMARK1 CMARK2 CMARK4}}
       {set mark3.marker-position {bit-or CMARK3 CMARK2 CMARK6}}
       {set mark7.marker-position {bit-or CMARK7 CMARK8 CMARK4}}
       {set mark9.marker-position {bit-or CMARK9 CMARK8 CMARK6}}
       }
      {else
       {set p1 {aref self.corners-array 0}}
       {set p2 {aref self.corners-array 1}}
       {set p3 {aref self.corners-array 2}}
       {set p4 {aref self.corners-array 3}}
       {set p1p2 {new Point {average p1.x p2.x} {average p1.y p2.y}}}
       {set p2p3 {new Point {average p2.x p3.x} {average p2.y p3.y}}}
       {set p3p4 {new Point {average p3.x p4.x} {average p3.y p4.y}}}
       {set p4p1 {new Point {average p4.x p1.x} {average p4.y p1.y}}}
       {mark2.set-position x=p1p2.x y=p1p2.y}
       {mark4.set-position x=p4p1.x y=p4p1.y}
       {mark6.set-position x=p2p3.x y=p2p3.y}
       {mark8.set-position x=p3p4.x y=p3p4.y}

       | now, are you wide/tall enough?  Presumes that parallel
       | sides are of the same length
       {set hedge-height {abs {- mark3.y mark1.y}}}
       {set hedge-width  {abs {- mark3.x mark1.x}}}
       {set vedge-height {abs {- mark9.y mark3.y}}}
       {set vedge-width  {abs {- mark9.x mark3.x}}}
       {cond {{or {> hedge-height {* 3 BOX-MARKER-HEIGHT}}
                  {> hedge-width  {* 3 BOX-MARKER-WIDTH}}}
              | we're wide enough
              {mark2.show}
              {mark8.show}
              {set mark1.marker-position
                   {bit-and mark1.marker-position {bit-not CMARK2}}}
              {set mark3.marker-position
                   {bit-and mark3.marker-position {bit-not CMARK2}}}
              {set mark7.marker-position
                   {bit-and mark7.marker-position {bit-not CMARK8}}}
```

```
                {set mark9.marker-position
                     {bit-and mark9.marker-position {bit-not CMARK8}}}
                }
               {else
                {mark2.hide}
                {mark8.hide}
                {set mark1.marker-position
                     {bit-or mark1.marker-position CMARK2}}
                {set mark3.marker-position
                     {bit-or mark3.marker-position CMARK2}}
                {set mark7.marker-position
                     {bit-or mark7.marker-position CMARK8}}
                {set mark9.marker-position
                     {bit-or mark9.marker-position CMARK8}}
                }} | else-cond
          {cond {{or {> vedge-height {* 3 BOX-MARKER-HEIGHT}}
                     {> vedge-width  {* 3 BOX-MARKER-WIDTH}}}
                | we're tall enough
                {mark4.show}
                {mark6.show}
                {set mark1.marker-position
                     {bit-and mark1.marker-position {bit-not CMARK4}}}
                {set mark3.marker-position
                     {bit-and mark3.marker-position {bit-not CMARK6}}}
                {set mark7.marker-position
                     {bit-and mark7.marker-position {bit-not CMARK4}}}
                {set mark9.marker-position
                     {bit-and mark9.marker-position {bit-not CMARK6}}}
                }
               {else
                {mark4.hide}
                {mark6.hide}
                {set mark1.marker-position
                     {bit-or mark1.marker-position CMARK4}}
                {set mark3.marker-position
                     {bit-or mark3.marker-position CMARK6}}
                {set mark7.marker-position
                     {bit-or mark7.marker-position CMARK4}}
                {set mark9.marker-position
                     {bit-or mark9.marker-position CMARK6}}
                }} | else-cond
         } | else
        } | cond
  } | method: move-markers
} | class: Canvas8Box
```

```
{define-class RotateBox {Canvas4Box}
  {define {init obj}
    obj-type:int=obj.marker-type
    {cond {{!= 0 {bit-and FREEROTATEABLE obj-type}}
           {super.init obj RotateBoxMarker}}
          {else
           {invoke-method CanvasBox 'init self obj}
           {set self.good-object-for-box false}
           }} | else-cond
    } | method: init
  } | class: RotateBox

{define-class ResizeBox {Canvas8Box}
  {define {init obj}
    obj-type:int=obj.marker-type
    {cond {{!= 0 {bit-and RESIZEABLE obj-type}}
           {super.init obj ResizeBoxMarker}
           }
          {else
           {super.init obj ResizeBoxMarker
                       extra-width=0 extra-height=0 border-width=10}
           {set self.good-object-for-box false}
           {self.hide-all-markers}
           }} | else-cond
    } | method: init
  } | class: ResizeBox
```

A.3.2 cobjects.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| Mixin class only.  Mixin class for the object that are put into the
| canvas.  Subclasses are currently in their own files.


{define {copy-object-properties from-obj:NewCanvasObject
                                 to-obj:NewCanvasObject
                                 add-properties:bool=false}:void
  prop-names:list
  prop-values:list

  {if {not {eq? {type->symbol {typeof from-obj}}
                {type->symbol {typeof to-obj}}}}
      {return}}
  {set prop-names
```

```
      {from-obj.canvas-object-properties.get-all-property-names}}
  {set prop-values
      {from-obj.canvas-object-properties.get-all-property-values}}
  {while {not {void? prop-names}}
    {if add-properties
        {to-obj.canvas-object-properties.set-one-property {car prop-names}
                                                          void}
        }
    {to-obj.change-object-property {car prop-names} {car prop-values}}
    {set prop-names {cdr prop-names}}
    {set prop-values {cdr prop-values}}
    } | while
  } | procedure: copy-object-properties


{define-mixin-class NewCanvasObject {Frame ImmitatingObject}
  | Base-class for an object that can be placed into the CanvasSheet.

  canvas:NewCanvasSheet
  graphic-external-width:int
  graphic-external-height:int
  internal-unrotated-left:int
  internal-unrotated-top:int
  internal-unrotated-width:int
  internal-unrotated-height:int
  rotation-angle:float | in radians
  | all coordinates are in the container's system
  object-internal-rotated-origin:Point  | in container coords
  object-center:Point                   | in container coords
  marker-type:int
  canvas-object-properties:CanvasPropertyList

  {define {init graphic-x-origin:int graphic-y-origin:int
               graphic-width:int graphic-height:int
               ...}
    {invoke-method ImmitatingObject 'init self}
    {generic-apply-method Frame 'init self
                          {rest-as-list}}
    {self.set-position x=graphic-x-origin y=graphic-y-origin}
    | store the width and height, since you can't get them easily from
    | Frame, and you want to use them for SelectBox'es, etc.
    {set self.graphic-external-width graphic-width}
    {set self.graphic-external-height graphic-height}
    {set self.rotation-angle 0}
    {self.graphic-resize graphic-x-origin
                         {+ graphic-x-origin graphic-width}
                         graphic-y-origin
```

```
                    {+ graphic-y-origin graphic-height}}
  {self.set-internal-corners
   {new Point graphic-x-origin graphic-y-origin}
   {new Point {+ graphic-x-origin graphic-width}
              {+ graphic-y-origin graphic-height}}}
  {set self.marker-type 0}  | should be re-set outside
  | need to add the properties that you know about
  | just set them to zero.  They should be initialized later.
  {set self.canvas-object-properties {new CanvasPropertyList}}
  {self.canvas-object-properties.set-one-property 'depth void}
  } | method: init


{define private {internal-initialize}:void
  {set self.object-center
       {new Point
            {+ self.x {/ self.graphic-external-width 2}}
            {+ self.y {/ self.graphic-external-height 2}}
            }}
  {invoke-method Frame 'set-property self
                 width=self.graphic-external-width
                 height=self.graphic-external-height}
  } | method: internal-initialize


{define {graphic-resize xmin:int xmax:int ymin:int ymax:int}:void
  width:int={+ {- xmax xmin} 1}
  height:int={+ {- ymax ymin} 1}
  old-x:int=self.x
  old-y:int=self.y
  {self.set-position x=xmin y=ymin}
  {set self.graphic-external-width width}
  {set self.graphic-external-height height}
  {self.set-property width=width height=height}
  {set self.internal-unrotated-left {+ self.internal-unrotated-left
                                       old-x
                                       {- xmin}
                                       }}
  {set self.internal-unrotated-top {+ self.internal-unrotated-top
                                      old-y
                                      {- ymin}
                                      }}
  {invoke-method NewCanvasObject 'internal-initialize self}
  } | method: graphic-resize


{define {set-internal-corners upper-left:Point lower-right:Point}:void
```

```
  | takes points in container coords.
  unrotated-upper-left:Point
  unrotated-lower-right:Point
  {set unrotated-upper-left
       {rotate-point upper-left
                     rad-angle={- self.rotation-angle}
                     center=self.object-center}}
  {set unrotated-lower-right
       {rotate-point lower-right
                     rad-angle={- self.rotation-angle}
                     center=self.object-center}}
  {set self.object-internal-rotated-origin upper-left}
  {set self.internal-unrotated-width
       {- unrotated-lower-right.x unrotated-upper-left.x}}
  {set self.internal-unrotated-height
       {- unrotated-lower-right.y unrotated-upper-left.y}}
  {set self.internal-unrotated-left {- unrotated-upper-left.x self.x}}
  {set self.internal-unrotated-top {- unrotated-upper-left.y self.y}}
  } | method: set-internal-corners


{define public {four-corners
                    rotated:bool=true
                    }:{return Point Point Point Point}
  | returns points in Container coords
  p1:Point
  p2:Point
  p3:Point
  p4:Point
  width-dx:int
  width-dy:int
  height-dx:int
  height-dy:int

  {cond {rotated
         {set width-dx
              {round {* self.internal-unrotated-width
                        {cosine self.rotation-angle}}}}
         {set width-dy
              {round {* self.internal-unrotated-width
                        {sine self.rotation-angle}}}}
         {set height-dx
              {round {* self.internal-unrotated-height
                        {sine self.rotation-angle}}}}
         {set height-dy
              {round {* self.internal-unrotated-height
                        {cosine self.rotation-angle}}}}
```

```
        | remember, though, that cartesian and pixel
        | coords are different (y is opposite)
        {set p1 {self.object-internal-rotated-origin.copy}}
        {set p2 {new Point {+ p1.x width-dx} {- p1.y width-dy}}}
        {set p3 {new Point {+ p2.x height-dx} {+ p2.y height-dy}}}
        {set p4 {new Point {- p3.x width-dx} {+ p3.y width-dy}}}
        } | rotated
       {else
        {set p1 {new Point self.x self.y}}
        {set p2 {new Point {+ p1.x self.graphic-external-width} p1.y}}
        {set p3 {new Point p2.x {+ p2.y self.graphic-external-height}}}
        {set p4 {new Point p1.x p3.y}}
        } | else
       } | cond
  {return p1 p2 p3 p4}
  } | method: four-corners


{define {encloses-object corner:Point diagcorner:Point}:bool
  overlap:symbol
  {set overlap {find-2D-rect-overlap
                corner diagcorner
                {new Point self.x self.y}
                {new Point
                     {+ self.x self.graphic-external-width}
                     {+ self.y self.graphic-external-height}}
                }}
  {return {or {eq? overlap OVERLAP-1SURROUNDS2}
              {eq? overlap OVERLAP-SAME}}}
  } | method: encloses-object


{define {partially-encloses-object corner:Point diagcorner:Point}:bool
  | assumes we're a  simple rectangle, possibly rotated
  corner1:Point
  corner2:Point
  corner3:Point
  corner4:Point
  p1:Point
  p2:Point
  overlap:symbol

  {cond {{= 0 self.rotation-angle}
         {set overlap {find-2D-rect-overlap
                       corner diagcorner
                       {new Point self.x self.y}
                       {new Point
```

```
                        {+ self.x self.graphic-external-width}
                        {+ self.y self.graphic-external-height}}
                    }}
        {if {eq? overlap OVERLAP-NONE}
            {return false}
            {return true}}
        }
       {else
        {set corner1 corner2 corner3 corner4
             {self.four-corners rotated=true}}
        {set overlap {find-2D-linerect-overlap corner1 corner2
                                               corner diagcorner}}
        {if {or {eq? overlap OVERLAP-PARTIAL}
                {eq? overlap OVERLAP-2SURROUNDS1}}
            {return true}}
        {set overlap {find-2D-linerect-overlap corner2 corner3
                                               corner diagcorner}}
        {if {or {eq? overlap OVERLAP-PARTIAL}
                {eq? overlap OVERLAP-2SURROUNDS1}}
            {return true}}
        {set overlap {find-2D-linerect-overlap corner3 corner4
                                               corner diagcorner}}
        {if {or {eq? overlap OVERLAP-PARTIAL}
                {eq? overlap OVERLAP-2SURROUNDS1}}
            {return true}}
        {set overlap {find-2D-linerect-overlap corner4 corner1
                                               corner diagcorner}}
        {if {or {eq? overlap OVERLAP-PARTIAL}
                {eq? overlap OVERLAP-2SURROUNDS1}}
            {return true}}
        }} | else-cond
  {return false}
  } | method: patially-encloses-object


{define {added-to-canvas canvas:NewCanvasSheet}:void
  {if {and {not {void? self.canvas}}
           {neq? canvas self.canvas}}
      {self.remove-from-canvas}
      }
  {set self.canvas canvas}
  } | method: added-to-canvas

{define {add-to-canvas canvas:NewCanvasSheet
                       copy-properties:bool=false
                       commit:bool=false}:void
  temp-value:any=self
```

```
  {self.added-to-canvas canvas}
  {when copy-properties
    {map {lambda {x}  | where x is a property name (symbol)
           {try
            {set temp-value {canvas.get-application-property x}}
            {catch {e:Exception}}
            }
           {try
            {set temp-value {canvas.get-canvas-property x}}
            {catch {e:Exception}}
            }
           {if {eq? self temp-value}
               {throw {new Exception
                           {format void
                                   "Unable to aquire property ~p" x}}}
               {self.change-object-property x temp-value}
               }
           } | lambda
         {self.canvas-object-properties.get-all-property-names}
         } | map
    } | when
  {canvas.exchange-canvas-objects void self commit=commit}
  } | method: add-to-canvas

{define {remove-from-canvas}:void
  {self.canvas.exchange-canvas-objects self void}
  } | method: remove-from-canvas

{define {canvas-point-to-local-coord p:Point unrotate:bool=false
                                     new-origin:bool=false}:Point
  | since object-center is in canvas coords, need to do this first
  {when unrotate
    {set p {rotate-point p rad-angle={- self.rotation-angle}
                         center=self.object-center}}
    }
  {set p {new Point {- p.x self.x} {- p.y self.y}}}
  {when new-origin
    {set p.x {- p.x self.internal-unrotated-left}}
    {set p.y {- p.y self.internal-unrotated-top}}
    } | when
  {return p}
  } | method: canvas-point-to-local-coord


{define {local-point-to-canvas-coord p:Point rotate:bool=false
                                     new-origin:bool=false}:Point
```

```
  | since object-center is in canvas coords, need to do this first
  local-p:Point={p.copy}
  {when new-origin
    | get it into object coordinates
    {set local-p.x {+ local-p.x self.internal-unrotated-left}}
    {set local-p.y {+ local-p.y self.internal-unrotated-top}}
    } | when
  | ok, now get it into canvas coordinates
  {set local-p.x {+ local-p.x self.x}}
  {set local-p.y {+ local-p.y self.y}}
  | finally, rotate it
  {if rotate
      {set local-p {rotate-point local-p rad-angle=self.rotation-angle
                                 center=self.object-center}}
      }
  {return local-p}
  } | method: local-point-to-canvas-coord


{define public {draw gc:GraphicContext}:void
  {return}
  } | method: draw

{define {reshape translator:{proc {Point}:Point}
                 p:Point=void
                 local-unrotated:bool=false
                 temporary:bool=false}:void
  {return}
  } | method: reshape

{define {rotate radian-angle:float temporary:bool=false}:void
  {error "Rotate not over-written"}
  } | method: rotate

{define {distance-to-object p:Point}:float
  {error "Failure to overwrite distance-to-object"}
  } | method: distance-to-object

{define {copy}:NewCanvasObject
  obj:NewCanvasObject
  {set obj {alloc {typeof self}}}
  {self.copy-guts obj}
  {return obj}
  } | method: copy

{define {copy-guts obj:NewCanvasObject}:void
  x-origin:int
```

```
  y-origin:int
  width:int
  height:int

  {set x-origin self.x}
  {set y-origin self.y}
  {set width self.graphic-external-width}
  {set height self.graphic-external-height}

  {invoke-method NewCanvasObject 'init obj
    x-origin y-origin width height}
  {obj.rotate self.rotation-angle}
  {copy-object-properties self obj add-properties=true}
  {set obj.marker-type self.marker-type}
  } | method: copy-guts


{define {change-object-property prop:symbol newval:any}:void
  {when {not {self.canvas-object-properties.has-property-local? prop}}
    {return}}
  {self.canvas-object-properties.set-one-property prop newval}
  {self.invalidate}
  } | method: change-object-property

{define {get-object-property prop:symbol default:any=void}:any
  {return {self.canvas-object-properties.get-property prop
                                                    default=default}}
  } | method: get-object-property

{define {save-to-file fout:OutputPort}:void
  | stored as:
  | <{type->symbol {typeof self}}>
  | <x-origin> <y-origin>
  | <internal-unrotated-width> <internal-unrotated-height>
  | <rotation-angle>
  | marker-type
  | begin-Properties
  | <prop-name1> <prop-value-type1> <prop-value1>
  | <prop-name2> <prop-value-type2> <prop-value2>
  | end-Properties
  prop-names:list
  prop-values:list
  prop-types:list

  {format fout "~p~%" {type->symbol {typeof self}}}
  {format fout "~d ~d~%" self.x self.y}
  {format fout "~d ~d~%"
```

```
        self.internal-unrotated-width self.internal-unrotated-height}
  {format fout "~p~%"    self.rotation-angle}
  {format fout "~d~%"    self.marker-type}
  {format fout "begin-Properties~%"}
  {set prop-names {self.canvas-object-properties.get-all-property-names}}
  {set prop-values
       {self.canvas-object-properties.get-all-property-values}}
  {set prop-types {map {lambda {x}  | where x is a prop-value
                         {return {type->symbol {typeof x}}}
                         }
                       prop-values}}
  {while {not {void? prop-names}}
    {format fout "~p ~p ~p~%"
            {car prop-names} {car prop-types} {car prop-values}}
    {set prop-names {cdr prop-names}}
    {set prop-values {cdr prop-values}}
    {set prop-types {cdr prop-types}}
    } | while
  {format fout "end-Properties~%"}
  } | method: save-to-file

{define {read-from-file fin:InputPort}:void
  temp-text:text
  input-list:list
  x-origin:int
  y-origin:int
  width:int
  height:int
  rotation-angle:float

  {set input-list {format-in fin "~s~d~d~d~d~f~d~s"}}
  {if {!= {list-length input-list} 8}
      {error {format void "Improper input file, bad ~s"
                           {type->symbol {typeof self}}}}}
  | get the items from the list
  | item 1
  {set temp-text {car input-list}}
  {set input-list {cdr input-list}}
  {if {not {eq? {text->symbol temp-text} {type->symbol {typeof self}}}}
      {error {format void "Improper input file, bad ~s"
                           {type->symbol {typeof self}}}}}
  | items 2 and 3
  {set x-origin {car input-list}}
  {set input-list {cdr input-list}}
  {set y-origin {car input-list}}
  {set input-list {cdr input-list}}
  | items 4 and 5
```

```
{set width {car input-list}}
{set input-list {cdr input-list}}
{set height {car input-list}}
{set input-list {cdr input-list}}

{invoke-method NewCanvasObject 'init self
  x-origin y-origin width height}

| item 6
{set rotation-angle {car input-list}}
{set input-list {cdr input-list}}
{self.rotate rotation-angle}

| item 7
{set self.marker-type {car input-list}}
{set input-list {cdr input-list}}

| item 8
{set temp-text {car input-list}}
{if {not {equal? temp-text "begin-Properties"}}
    {error {format void "Improper input file, no object properties"}}}
{loop
 prop-name:any
 prop-type:any
 prop-value:any

  {set prop-name {read-string fin}}
  {if {void? prop-name}
      {error "Improper input file, no end-Properties"}}
  {if {equal? prop-name "end-Properties"}
      {break}}
  {set prop-name {text->symbol prop-name}}

  {set prop-type {read-string fin}}
  {if {void? prop-type}
      {error {format void "~p~p"
                     "Improper input file, "
                     "premature end while expecting property type"}}}
  {set prop-type {text->symbol prop-type}}
  {cond {{eq? prop-type {type->symbol int}}
         {set prop-value {read-int fin}}
         }
        {{eq? prop-type {type->symbol text}}
         {set prop-value {read-string fin}}
         }
        {{eq? prop-type {type->symbol symbol}}
         {set prop-value {read-string fin}}
```

```
            {if {not {void? prop-value}}
                {set prop-value {text->symbol prop-value}}}
            }
           {{eq? prop-type {type->symbol float}}
            {set prop-value {read-float fin}}
            }
           {else
            {error {format void "~p~p"
                           "Improper input file, unknown type "
                           prop-type}}
            }} | else-cond
      {if {void? prop-value}
          {error {format void "~p~p"
                         "Improper input file, no value for property "
                         prop-name}}
          }
      {self.canvas-object-properties.set-one-property prop-name prop-value}
      } | loop
    } | method: read-from-file

  } | mixin-class: NewCanvasObject


{define-mixin-class NewCanvasDrawingObject {NewCanvasObject}
  | Base-class for an objects that export to just the "draw" method
  | (Lines, rectangles, circles, etc.)

  {define {init ...}
    {myapply-method NewCanvasObject 'init self {rest-as-list}}
    } | method: init


  {define {export-draw fout:OutputPort}:void
    {return}
    } | method: export-draw
  } | mixin-class: NewCanvasObject
```

A.3.3 line.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| The event handler and the CanvasObject for representing
| line-objects.  This includes single-line, poly-line, polygon,
| and regular-polygon.

{define-class LineModeEventHandler {NewCanvasDrawMode}
```

```
creating:bool    | if we are in the process of creating a poly-line
last-press-p:Point      | where the last mouse press is
last-pos-p:Point    | where the last sighting of the mouse was
points:list      | points are in reverse order

{define {init}
  | set up the points so that we can mutate them and save memory
  {set self.last-press-p {new Point 0 0}}
  {set self.last-pos-p {new Point 0 0}}
  {return}
  } | method: init

{define {toggle-creating}:void
  {set self.creating {not self.creating}}
  {self.new-descriptions}
  {if self.creating
      {self.canvas.lock-semaphore 'mode}
      {self.canvas.release-semaphore 'mode}
      }
  } | method: toggle-creating

{define private {quit-and-restart save-line:bool=true}:void
  obj:CanvasLineObject
  e:CanvasKeyEvent

  | get the new line all set up
  {cond {save-line
         {set self.points {reverse self.points}}
         {set obj {new CanvasLineObject self.points}}
         {obj.add-to-canvas self.canvas.canvas.Sheet
                            copy-properties=true commit=true}
         {self.toggle-creating}
         } | cond saving the line
        {else
         | now clean up yourself
         | bugbug: Doesn't do it right if a "draw" was
         | called in the middle.
         | don't both toggle-creating, it's done with the last backspace
         {while {not {void? self.points}}
           {set e {new CanvasKeyEvent}}
           {set e.key 127}
           {self.key-press e}
           } | while
         }} | else-cond
  | don't bother setting last-press-x/y, they won't be used
  | do release self.points, just for GarbageCollection
  {set self.points void}
```

```
  } | method: quit-and-restart


{define {new-descriptions}:void
  {if self.creating
      {self.canvas.set-event-descriptions
       pointer-press="Next point"
       pointer-dbl-clk="Add final point"
       shiftify="Horiz/Vert"
       }
      {self.canvas.set-event-descriptions
       pointer-press="Start Polyline"
       pointer-dbl-clk="Draw a point"
       }
      }
  } | method: new-descriptions

{define {pointer-motion e:PointerEvent snap-point:Point}:void
  x:int   | for easy change: this is the point used
  y:int   | for easy change: this is the point used

  {when self.creating
    | Currently drawing a polyline, do the rubberband-line
    | thing to show where the next line would go.

    | first erase the last rubberband-line
    {if {not self.invalidated}
        {rubberband-line self.canvas.canvas.Sheet
                         self.last-press-p.x self.last-press-p.y
                         self.last-pos-p.x self.last-pos-p.y}
        {set self.invalidated false}
        }
    | now update
    {set self.last-pos-p.x snap-point.x}
    {set self.last-pos-p.y snap-point.y}
    | and finally, draw the new rubberband-line
    {rubberband-line self.canvas.canvas.Sheet
                     self.last-press-p.x self.last-press-p.y
                     self.last-pos-p.x self.last-pos-p.y}
    } | when
  } | method: pointer-motion


{define {pointer-press e:PointerEvent snap-point:Point}:void
  | Places the next/first point for a poly-line
  {cond {{not self.creating}
         | we need to start creating.
```

```
         {self.toggle-creating}
         {set self.points {new list snap-point void}}
         {set self.last-press-p.x snap-point.x}
         {set self.last-press-p.y snap-point.y}
         {set self.last-pos-p.x snap-point.x}
         {set self.last-pos-p.y snap-point.y}
         } | cond not-creating
        {else
         | we need to add a point and update everything
         {set self.points {cons snap-point self.points}}
         {set self.last-press-p.x snap-point.x}
         {set self.last-press-p.y snap-point.y}
         {set self.last-pos-p.x snap-point.x}
         {set self.last-pos-p.y snap-point.y}
         } | else (creating)
        } | cond
  } | method: pointer-press

{define {pointer-dbl-clk e:PointerEvent snap-point:Point}:void
  | already got the first click on this point
  | so, all we have to do it close up shop
  {self.quit-and-restart}
  } | method: pointer-dbl-clk

{define {key-press e:KeyEvent}:void
  {cond {{not self.creating}
         {return}
         }
        {{or {= e.key 13} {= e.key 10}}   | return
         {self.quit-and-restart}
         }
        {{or {= e.key 127} {= e.key 8}}   | backspace/delete
         {when self.creating
           | erase lines.
           {rubberband-line self.canvas.canvas.Sheet
                            self.last-press-p.x self.last-press-p.y
                            self.last-pos-p.x self.last-pos-p.y}
           {set self.points {cdr self.points}}
           {cond {{void? self.points}
                  {self.toggle-creating}
                  }
                 {else
                  {rubberband-line
                     self.canvas.canvas.Sheet
                     {car self.points}.x {car self.points}.y
                     self.last-press-p.x self.last-press-p.y}
                  {set self.last-press-p.x {car self.points}.x}
```

```
                {set self.last-press-p.y {car self.points}.y}
                | make a new line
                {rubberband-line
                   self.canvas.canvas.Sheet
                   self.last-press-p.x self.last-press-p.y
                   self.last-pos-p.x self.last-pos-p.y}
                } | else
               } | cond
          } | when
         } | backspace or delete
        {else
         | ignore
         {return}
         } | else
        } | cond
  } | method: key-press

{define {cancel}:void
  {if self.creating
      {self.quit-and-restart save-line=false}
      }
  } | method: cancel

{define {draw gc:GraphicContext}:void
  point1:Point
  point2:Point
  restpoints:list

  | If currently drawing a line, draws the finished line segments.
  {cond {{not self.creating}
         {return}
         }
        {{= {list-length self.points} 1}  | just a point
         {set point1 {car self.points}}
         | {rubberband-line self.canvas.canvas.Sheet
         | point1.x point1.y {+ 1 point1.x} {+ 1 point1.y}}
         {gc.draw-line point1.x point1.y
                       {+ 1 point1.x} {+ 1 point1.y}
                       'black 1}
         } | just a point
        {else
         | bugbug: Does NOT do the rubberband-line
         |         for the current segment.
         | (That ended up usually being not currently possible in curl.)
         {set point2 {car self.points}}
         {set restpoints {cdr self.points}}
         {while {not {void? restpoints}}
```

```
            {set point1 point2}
            {set point2 {car restpoints}}
            {set restpoints {cdr restpoints}}
            | {rubberband-line self.canvas.canvas.Sheet
            | point1.x point1.y point2.x point2.y}
            {gc.draw-line point1.x point1.y point2.x point2.y 'black 1}
            } | while
          } | else
        } | cond
    | {rubberband-line self.canvas.canvas.Sheet
    |             self.last-press-p.x self.last-press-p.y
    |             self.last-pos-p.x self.last-pos-p.y}
    {gc.draw-line self.last-press-p.x self.last-press-p.y
                  self.last-pos-p.x self.last-pos-p.y 'black 1}

    } | method: draw

  {define {selected}:void
    {self.canvas.set-message t={format void "Polyline mode selected"}}
    {self.new-descriptions}
    } | method: selected

  {define {popup-dialog-box}:void
    | bugbug: ck: actually do this sometime
    {super.popup-dialog-box}
    } | method: popup-dialog-box

  } | class: LineModeEventHandler

|#####################################################################
|#####################################################################
|#####################################################################
|#####################################################################

{define-class CanvasLineObject {NewCanvasDrawingObject}
  container-rotated-points:list | where the canvas sees them drawn as

  {define {init points:list}
    minx:int
    maxx:int
    miny:int
    maxy:int

    {set self.container-rotated-points points}
    | dummy call to the super.  We'll resize it soon enough...
    {super.init 0 0 0 0}
    {self.internal-initialize}
```

```
  | don't forget the marker-type
  {set self.marker-type {bit-or self.marker-type
                                RESIZEABLE
                                FREEROTATEABLE}}
  | and now our special properties
  {self.canvas-object-properties.set-one-property 'line-width void}
  {self.canvas-object-properties.set-one-property 'line-color void}
  } | method: init

{define private {internal-initialize}:void
  minx:int
  maxx:int
  miny:int
  maxy:int
  point1:Point
  point2:Point
  local-unrotated-points:list

  | first, the resize thing
  {set minx maxx miny maxy {points-minmax self.container-rotated-points}}
  {self.graphic-resize minx maxx miny maxy}
  | copy over to local-unrotated-points.
  | Can't do it before resizing the box.
  {set local-unrotated-points
       {map {lambda {x}
              {return {self.canvas-point-to-local-coord x unrotate=true}}
              }
            self.container-rotated-points}}
  | now the rotated resize thing
  {cond {{= self.rotation-angle 0}
         {self.set-internal-corners {new Point minx miny}
                                    {new Point maxx maxy}}
         }
        {else
         {set minx maxx miny maxy {points-minmax local-unrotated-points}}
         {set point1
              {self.local-point-to-canvas-coord {new Point minx miny}
                                                rotate=true}}
         {set point2
              {self.local-point-to-canvas-coord {new Point maxx maxy}
                                                rotate=true}}
         {self.set-internal-corners point1 point2}
         }} | else-cond
  | now the marker-type thing
  {if {= {list-length self.container-rotated-points} 2}
      {set self.marker-type {bit-or self.marker-type 2POINTOBJECT}}
      }
```

```
  } | method: internal-initialize


{define public {re-initialize}:void
  {self.internal-initialize}
  {self.invalidate}
  } | method: re-initialize


{define public {four-corners
                  rotated:bool=true
                  }:{return Point Point Point Point}
  p1:Point
  p2:Point
  p3:Point
  p4:Point
  p-temp:Point

  {set p1 p2 p3 p4 {super.four-corners rotated=rotated}}
  {if {or {= 0 {bit-and self.marker-type 2POINTOBJECT}}
          {not rotated}}
      {return p1 p2 p3 p4}}
  | We need to figure out if we're returning p1 and p3, or p2 and p4...
  {set p-temp {car self.container-rotated-points}}
  {if {or {and {= p-temp.x p1.x} {= p-temp.y p1.y}}
          {and {= p-temp.x p3.x} {= p-temp.y p3.y}}}
      {return p1 p3 void void}
      {return p2 p4 void void}
      }
  } | method: four-corners


{define {partially-encloses-object corner:Point diagcorner:Point}:bool
  | points in canvas coords
  temp-points:list
  corner1:Point
  corner2:Point
  corner3:Point
  corner4:Point
  p1:Point
  p2:Point
  overlap:symbol

  {if {void? temp-points}
      | we only have one point, no "lines"
      {return {super.partially-encloses-object corner diagcorner}}}
```

```
  | it's easier if we know which points we have
  {set corner1 corner3 {twocorners->origdiag corner diagcorner}}
  {set corner2 {new Point corner3.x corner1.y}}
  {set corner4 {new Point corner1.x corner3.y}}

  | initialize for cycling through the points.
  {set temp-points self.container-rotated-points}
  | hang on, do we only have one point?
  {if {void? temp-points}
      {return {super.partially-encloses-object corner1 corner3}}}
  | ok, do this by each line segment
  {set p2 {car temp-points}}
  {set temp-points {cdr temp-points}}
  {while {not {void? temp-points}}
    {set p1 p2}
    {set p2 {car temp-points}}
    {set temp-points {cdr temp-points}}
    {set overlap {find-2D-linerect-overlap p1 p2 corner diagcorner}}
    {cond {{eq? overlap OVERLAP-NONE}
           {continue}}
          {{eq? overlap OVERLAP-2SURROUNDS1}
           {return true}}
          {else | overlap-partial
           {return true}
           } | else
          } | cond
    } | while
  } | method: partially-encloses-object


{define public {draw gc:GraphicContext}:void
  | standard draw function.  Just uses gc.draw-line
  | bugbug: only obeys line-color and line-width.
  point1:Point
  point2:Point
  temppoints:list
  line-color:any={self.get-object-property 'line-color}
  line-width:int={self.get-object-property 'line-width}

  {set temppoints
       {map {lambda {x}
              {return {self.canvas-point-to-local-coord x}}}
            self.container-rotated-points}}
  {cond {{= {list-length temppoints} 1}   | just a point
         {set point1 {car temppoints}}
         {gc.draw-line point1.x point1.y
                       {+ 1 point1.x} {+ 1 point1.y}
```

```
                    line-color line-width}
        } | just one point
       {else
        {set point2 {car temppoints}}
        {set temppoints {cdr temppoints}}
        {while {not {void? temppoints}}
          {set point1 point2}
          {set point2 {car temppoints}}
          {set temppoints {cdr temppoints}}
          {gc.draw-line point1.x point1.y point2.x point2.y
                        line-color line-width}
          } | while
        } | else
       } | cond
  } | method: draw


{define {reshape translator:{proc {Point}:Point} p:Point=void
                local-unrotated:bool=false
                temporary:bool=false}:void
  input-list:list
  temp-list:list
  old-p:Point
  new-point:Point
  new-points:list

  {if {and local-unrotated {not {void? p}}}
      {throw {new Exception
                  {format void "~p~p"
                          "Trying to move a single point in an "
                          "unrotated-world.  Not allowed"}}}}

  {if {not local-unrotated}
      {set input-list self.container-rotated-points}
      {set input-list {map {lambda {x}
                             {return {self.canvas-point-to-local-coord
                                      x
                                      unrotate=true
                                      new-origin=true}}
                             }
                           self.container-rotated-points}}
      } | if

  {cond {{void? p}
         {set new-points {map {lambda {x}
                                {return {translator x}}}
                              input-list}}
```

```
      } | cond all the points
     {else
      | just one point moving
      | It would be faster if I used the first point that
      | matched, but this feels a bit more right...
      {set temp-list {filter {lambda {temp-p}:bool
                               {return {and {= temp-p.x p.x}
                                            {= temp-p.y p.y}}}
                               } | lambda
                             input-list}}
      {if {void? temp-list}
          {error "Line.reshape: Moving non-existent point"}}
      | just pick one of the points.  Last seems to make most
      | sense to me.
      {set old-p {last temp-list}}
      {cond
       {temporary
        | need to remember three points to draw the
        | rubberband-line
        {cond
         {{eq? {car input-list} old-p}
          {set new-points void}
          {set temp-list input-list}
          }
         {else
          {set temp-list
               {cdr-down-by-proc {lambda {x:list}:bool
                                   {return {and {not {void? {cdr x}}}
                                                {eq? {cadr x} old-p}}}
                                   }
                                 input-list}}
          {set new-points {make-list {car temp-list}}}
          {set temp-list {cdr temp-list}}
          }
         } | cond
        } | cond-temporary
       {else
        | we have to be in rotated-land.
        {set new-point {translator old-p}}
        }} | else-cond
       }} | cond

{if local-unrotated
    | need to rotate the points
    {set new-points
         {map {lambda {x}
                {return
```

```
                {self.local-point-to-canvas-coord x rotate=true
                                                  new-origin=true}}}
              new-points}}
    }

  {cond {temporary
         {rubberband-lines self.canvas new-points}
         }
        {{void? p}
         {set self.container-rotated-points new-points}
         {self.re-initialize}
         }
        {else
         {set old-p.x new-point.x}
         {set old-p.y new-point.y}
         }
        } | cond
  } | method: reshape



{define {rotate radian-angle:float temporary:bool=false}:void
  new-points:list
  {set new-points
       {map {lambda {x}
              {return
               {rotate-point x
                             rad-angle=radian-angle
                             center=self.object-center}}
              }
             self.container-rotated-points}}
  {cond {temporary
         {rubberband-lines self.canvas new-points}
         }
        {else
         {set self.container-rotated-points new-points}
         {set self.rotation-angle {+ self.rotation-angle radian-angle}}
         {self.re-initialize}
         }
        }
  } | method: rotate


{define {distance-to-object p:Point}:float
  point1:Point
  point2:Point
  restpoints:list
```

```
  min-dist:float

  {if {= {list-length self.container-rotated-points} 1}
      {return {two-point-distance p {car self.container-rotated-points}}}
      }

  | we have to look at each of the line segments and find the min
  {set point2 {car self.container-rotated-points}}
  {set restpoints {cdr self.container-rotated-points}}
  | hack: use MAXINT/2 because "min" sometimes does weird stuff when
  | passed a variable of MAXINT size.
  {set min-dist {/ MAXINT 2}}
  {while {not {void? restpoints}}
    {set point1 point2}
    {set point2 {car restpoints}}
    {set restpoints {cdr restpoints}}
    {set min-dist
         {min min-dist {point-to-lineseg-distance p point1 point2}}}
    } | while
  {return min-dist}
  } | method: distance-to-object


{define {copy-guts obj:NewCanvasObject}:void
  obj-local:CanvasLineObject=obj   | for type checking
  | points are rotated by the super.
  {set obj-local.container-rotated-points
       {map {lambda {x}  | where x is a Point
              {return {rotate-point x rad-angle={- self.rotation-angle}
                                    center=self.object-center}}
              }
            self.container-rotated-points}}
  {super.copy-guts obj}
  } | method: copy-guts


{define {save-to-file fout:OutputPort}:void
  | stored as (points in Canvas coordinates, but unrotated):
  | <{type->symbol {typeof self}}>
  | begin-Points
  | <px1> <py1>
  | <px2> <py2>
  | end-Points
  | <<<super.save-to-file>>>
  temp-points:list

  {format fout "~p~%" {type->symbol {typeof self}}}
```

```
  {format fout "begin-Points~%"}
  {set temp-points
       {map {lambda {x}
              {return {rotate-point x rad-angle={- self.rotation-angle}
                                    center=self.object-center}}}
            self.container-rotated-points}}
  {while {not {void? temp-points}}
    {format fout "~d ~d~%" {car temp-points}.x {car temp-points}.y}
    {set temp-points {cdr temp-points}}
    } | while
  {format fout "end-Points~%"}
  {super.save-to-file fout}
  } | method: save-to-file

{define {read-from-file fin:InputPort}:void
  | the "new-object" line is already read
  input-list:list
  temp-text:text
  temp-points:list

  {set input-list {format-in fin "~s~s"}}
  {if {!= {list-length input-list} 2}
      {error {format void "Improper input file, bad ~s"
                     {type->symbol {typeof self}}}}}
  | get the items from the list
  | item 1
  {set temp-text {car input-list}}
  {set input-list {cdr input-list}}
  {if {not {eq? {text->symbol temp-text} {type->symbol {typeof self}}}}
      {error {format void "Improper input file, bad ~s"
                     {type->symbol {typeof self}}}}
      {set temp-text {car input-list}}
      }

  {set temp-text {car input-list}}
  {if {not {equal? temp-text "begin-Points"}}
      {error {format void "Improper input file, no line points"}}
      }
  {loop
   x:any
   y:any

   {set x {read-int fin}}
   {when {void? x}
     {set temp-text {read-string fin}}
     {if {and {not {void? temp-text}}
              {equal? "end-Points" temp-text}}
```

```
        {break}
        {error {format void "Improper input file, bad points."}}}
    } | when
   {set y {read-int fin}}
   {if {void? y}
       {error {format void "Improper input file, bad points."}}}
   {set temp-points {cons {new Point x y} temp-points}}
   } | loop
  | Pretend that you're init.  Don't want to call it, because
  | then super would be called.
  {set self.container-rotated-points {reverse temp-points}}
  | The following is like "super.init"
  {super.read-from-file fin}
  {self.internal-initialize}
  } | method: read-from-file


{define {export-draw fout:OutputPort}:void
  | export in container coords, which may be group coords
  point1:Point
  point2:Point
  restpoints:list

  {if {eq? {type->symbol {typeof self}} {type->symbol CanvasLineObject}}
      {format fout "    | CanvasLineObject stuff~%"}
      }
  {format fout "    {let~%"}
  {format fout "      line-color:any=~p~%"
          {color->text {self.get-object-property 'line-color}
                       quote-sym=true}}
  {format fout "      line-width:int=~d~%"
          {self.get-object-property 'line-width}}
  {cond {{= {list-length self.container-rotated-points} 1}
         | just a point
         {set point1 {car self.container-rotated-points}}
         {format fout "~p~d ~d ~d ~d~p"
                 "      {gc.draw-line "
                 point1.x point1.y {+ 1 point1.x} {+ 1 point1.y}
                 " line-color line-width}"}
         } | just one point
        {else
         {set point2 {car self.container-rotated-points}}
         {set restpoints {cdr self.container-rotated-points}}
         {while {not {void? restpoints}}
           {set point1 point2}
           {set point2 {car restpoints}}
           {set restpoints {cdr restpoints}}
```

```
            {format fout "~p~d ~d ~d ~d~p~%"
                    "      {gc.draw-line "
                    point1.x point1.y point2.x point2.y
                    " line-color line-width}"}
            } | while
          }} | else and cond
    {format fout "    } | let~%"}
    {super.export-draw fout}
    {format fout "~%"}
    } | method: export-draw
  } | class:CanvasLineObject
```

A.3.4 markers.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This contains the stuff for the little black select markers.  (Both
| for vertex-es and for selection boxes.)

{define-mixin-class NewCanvasSelectMarker {CanvasDragee SimpleGraphic}
  | One of those little black squares that signifies a selected
  | object.
  | coordinates are in the container's system, whatever container that
  | may be.

  | appearance stuff
  public center:Point
  public width:int
  public height:int
  private color:any
  private shape-style:symbol
  public showing:bool

  | internal information
  protected dragging:bool
  protected just-selected:bool  | For the first pointer-motion
  private start-p:Point
  private last-p:Point
  public marker-position:int  | where in an 8box, or a vertex

  | and, related objects
  protected canvas:NewCanvasSheet
  public box:CanvasBox
  public obj:NewCanvasObject

  {define {init x-center:int y-center:int
```

```
              marker-pos:int
              width:int=BOX-MARKER-WIDTH
              height:int=BOX-MARKER-HEIGHT
              color:any=BOX-MARKER-COLOR
              shape-style:symbol=FILLEDBOX-MARKER-SHAPE}
  sm-half-width:int={/ width 2}
  lg-half-width:int={+ sm-half-width {% width 2}}
  sm-half-height:int={/ height 2}
  lg-half-height:int={+ sm-half-height {% height 2}}

  {invoke-method SimpleGraphic 'init self}
  | Hack: can't get at dragee's init, but it doesn't do anything good.
  {invoke-method CanvasDragee 'init self}
  {self.set-position x={- x-center sm-half-width}
                     y={- y-center sm-half-height}}

  {set self.center {new Point x-center y-center}}
  {set self.width width}
  {set self.height height}
  {set self.color color}
  {set self.shape-style shape-style}
  {set self.showing true}

  {set self.start-p {new Point 0 0}}
  {set self.last-p {new Point 0 0}}
  {set self.marker-position marker-pos}
  } | method: init

{define {set-canvas canvas:NewCanvasSheet}:void
  {set self.canvas canvas}
  {if self.showing
      {canvas.append-object self}}
  } | methdo: set-canvas

{define {show}:void
  {set self.showing true}
  {self.canvas.append-object self}
  } | method: show

{define {hide}:void
  {when self.showing
    {set self.showing false}
    {self.canvas.remove-object self}
    } | when
  } | method:hide

{define {set-box box:CanvasBox}:void
```

```
  {set self.box box}
  {set self.obj box.obj}
  {self.set-canvas box.canvas}
  } | method: set-box

{define {localcoords->canvascoords x:int y:int}:{return int int}
  {return {+ x self.x} {+ y self.y}}}
{define {canvascoords->localcoords x:int y:int}:{return int int}
  {return {- x self.x} {- y self.y}}}


{define {get-marker-offsets rotated:bool=true}:{return int int}
  angle:float
  p:Point={new Point
              {- self.last-p.x self.start-p.x}
              {- self.last-p.y self.start-p.y}}
  {when {and rotated self.box.rotated}
    {set angle {- self.box.obj.rotation-angle}}
    {set p {rotate-point p rad-angle=angle}}
    } | when
  {return p.x p.y}
  } | method: get-marker-offsets


{define {get-similar-boxes}:list
  | assume that nobody is similar.  probably want to OVERWRITE
  {return {make-list self.box}}
  } | method: get-similar-boxes


{define {update-dragging e:PointerEvent
                         snap-p:Point
                         current-select-boxes:list}:void
  | called only when selected and getting pointer-motion events.
  | p in NewCanvasSheet coordinates
  | OVERWRITE
  {return}
  }

{define {done-dragging current-select-boxes:list}:void
  {return}
  }

| ***** methods for Dragee ***** |

{define public {get-drag-image}:Graphic
  | a bit of a hack here.  I don't know how else to get this information
```

```
    {set self.dragging true}
    {return {invoke-method Dragee 'get-drag-image self}}
    }

  | ***** methods for Dragee's super, Selectee ***** |

  {define public {on-select}:void
    {super.on-select}
|    {self.obj.hide}
|    {self.obj.temp-reshape {lambda {x:Point}:Point {return x}}}
    {set self.just-selected true}
    } | method: on-select

  {define public {on-deselect}:void
    {super.on-deselect}
|    {self.obj.show}
    {set self.just-selected false}
    {when self.dragging
      {self.done-dragging {self.get-similar-boxes}}
      {set self.dragging false}
      } | when
    } | method: on-deselect

  | ignore pointer-press and pointer-release, since (with my little
  | addition below) those will cause on-select and on-deselect.
  | (Which would be the proper thing to overwrite.)  Unless you feel
  | the need to pass the pointer-event elsewhere and not be selected.
  | (But I think I've designed this so you shouldn't feel that need.

  {define public {pointer-press e:PointerEvent}:void
    {set self.start-p.x e.x}
    {set self.start-p.y e.y}
    {invoke-method Dragee 'pointer-press self e}
    }

  {define public {pointer-motion e:PointerEvent}:void
    snap-p:Point

    {invoke-method Dragee 'pointer-motion self e}
    | temp conversion now to Canvas coords
    {set e.x e.y {self.localcoords->canvascoords e.x e.y}}
    {set snap-p {self.canvas.WindowEvent2snap e}}
    {self.canvas.pointer-motion e}
    | and convert back, including the snap-p
    {set e.x e.y {self.canvascoords->localcoords e.x e.y}}
    {set snap-p.x snap-p.y {self.canvascoords->localcoords e.x e.y}}
```

```
  {set snap-p {new Point e.x e.y}}
  {when self.dragging
    {set self.last-p.x e.x}
    {set self.last-p.y e.y}
    {self.update-dragging e snap-p {self.get-similar-boxes}}
    } | when
  } | method: pointer-motion

{define public {pointer-release e:PointerEvent}:void
  {invoke-method Dragee 'pointer-release self e}
  {self.deselect}
  } | method: pointer-release

{define public {enter e:WindowEvent}:void
  {invoke-method Dragee 'enter self e}
  {when {not self.dragging}
    {set e.x e.y {self.localcoords->canvascoords e.x e.y}}
    | make the canvas believe that we never left
    {self.canvas.enter e}
    } | when
  } | method: enter

{define public {leave e:WindowEvent}:void
  {invoke-method Dragee 'leave self e}
  {when {not self.dragging}
    {set e.x e.y {self.localcoords->canvascoords e.x e.y}}
    | better "leave" the canvas, else it might be confused.
    {self.canvas.leave e}
    } | when
  } | method: leave

| ***** methods for SimpleGraphic ***** |


{define public {layout}:{return int int}
  | Necessary for super SimpleGraphic to work
  {return self.width self.height}
  } | method: layout

{define public {set-position x:any=void y:any=void}:void
  {when {not {void? x}}
    {set self.x x}
    {set x {- x {/ self.width 2}}}
    }
  {when {not {void? y}}
    {set self.y y}
    {set y {- y {/ self.width 2}}}
```

```
    }
  {super.set-position x=x y=y}
  } | method: set-position

{define public {draw gc:GraphicContext}:void
  | Standard draw method of a Graphic object.
  right-side:int={- self.width 1}
  bottom-side:int={- self.height 1}

  {super.draw gc}
  {cond {{eq? self.shape-style HOLLOWBOX-MARKER-SHAPE}
         {gc.fill-rect 0 0 right-side bottom-side
                       {self.get-property 'background}}
         {gc.draw-line 0 0 right-side 0 self.color 1}
         {gc.draw-line right-side 0 right-side bottom-side self.color 1}
         {gc.draw-line right-side bottom-side 0 bottom-side self.color 1}
         {gc.draw-line 0 bottom-side 0 0 self.color 1}
         }
        {{eq? self.shape-style FILLEDBOX-MARKER-SHAPE}
         {gc.fill-rect 0 0 right-side bottom-side self.color}
         }
        {{eq? self.shape-style HOLLOWCIRCLE-MARKER-SHAPE}
         {gc.draw-ellipse 0 0 right-side bottom-side void self.color 1}
         }
        {{eq? self.shape-style FILLEDCIRCLE-MARKER-SHAPE}
         {gc.draw-ellipse 0 0 right-side bottom-side
                          self.color self.color 1}
         }
        {else
         {error {format void "Unknown shape-style ~p" self.shape-style}}
         } | else
        } | cond
  } | method: draw

} | mixin: NewCanvasSelectMarker



{define-mixin-class BoxMarker {NewCanvasSelectMarker}
  {define {init ...}
    {myapply-method NewCanvasSelectMarker 'init self {rest-as-list}}
    } | method: init

  {define {get-similar-boxes}:list
    boxes:list
    {set boxes self.canvas.select-boxes}
    {set boxes {filter {lambda {x}:bool
```

```
                        {return x.good-object-for-box}}
                       boxes}}
    {return boxes}
    } | method: get-similar-boxes
  } | class: BoxMarker


{define-class ResizeBoxMarker {BoxMarker}
  scale-x:float
  scale-y:float
  changing-object:NewCanvasObject
  translator:{proc {Point}:Point}

  {define {init x-center:int y-center:int marker-pos:int ...}
    {generic-apply-method NewCanvasSelectMarker 'init self
                          {rest-as-list}
                          x-center y-center marker-pos
                          shape-style=HOLLOWBOX-MARKER-SHAPE}

    | known types are: CMARK1, CMARK2, CMARK3, CMARK4, CMARK6,
    | CMARK7, CMARK8, CMARK9
    {cond {{!= 0 {bit-and marker-pos
                          {bit-or CMARK1 CMARK2 CMARK3
                                  CMARK4 CMARK6
                                  CMARK7 CMARK8 CMARK9}}}
          {set self.translator
               {lambda {p:Point}:Point
                 x:int
                 y:int
                 top-edge:int
                 right-edge:int
                 bottom-edge:int
                 left-edge:int

                 {cond
                  {self.box.rotated
                   {set top-edge 0}
                   {set left-edge 0}
                   {set right-edge
                        self.changing-object.internal-unrotated-width}
                   {set bottom-edge
                        self.changing-object.internal-unrotated-height}
                   }
                  {else
                   {set top-edge self.changing-object.y}
                   {set left-edge self.changing-object.x}
                   {set right-edge
```

```
      {+ self.x
         self.changing-object.graphic-external-width}}
  {set bottom-edge
      {+ self.y
         self.changing-object.graphic-external-height}}
  }} | else-cond
{cond {{or {!= 0 {bit-and marker-pos CMARK1}}
           {!= 0 {bit-and marker-pos CMARK4}}
           {!= 0 {bit-and marker-pos CMARK7}}
           }
      {set x {+ {* {- p.x right-edge}
                   self.scale-x}
                right-edge}}
      }
     {{or {!= 0 {bit-and marker-pos CMARK3}}
          {!= 0 {bit-and marker-pos CMARK6}}
          {!= 0 {bit-and marker-pos CMARK9}}
          }
      {set x {+ {* {- p.x left-edge}
                   self.scale-x}
                left-edge}}
      }
     {else
      {set x p.x}
      }} | else-cond

{cond {{or {!= 0 {bit-and marker-pos CMARK1}}
           {!= 0 {bit-and marker-pos CMARK2}}
           {!= 0 {bit-and marker-pos CMARK3}}
           }
      {set y {+ {* {- p.y bottom-edge}
                   self.scale-y}
                bottom-edge}}
      }
     {{or {!= 0 {bit-and marker-pos CMARK7}}
          {!= 0 {bit-and marker-pos CMARK8}}
          {!= 0 {bit-and marker-pos CMARK9}}
          }
      {set y {+ {* {- p.y top-edge}
                   self.scale-y}
                top-edge}}
      }
     {else
      {set y p.y}
      }} | else-cond
{return {new Point x y}}
} | lambda
```

```
            } | set
         } | cond-CMARK1-9
        {else
         {error {format void "~p~s~p"
                        "ResizeBoxMarker: Bad marker position \""
                        marker-pos
                        "\""}}
         } | else
        } | cond
  } | method: init


{define {reshape-objects-in-boxes boxes:list temporary=false}:void
  local-unrotated:bool=self.box.rotated
  {map {lambda {x}
         {set self.changing-object x.obj}
         {if temporary
             {x.obj.reshape self.translator
                            local-unrotated=local-unrotated
                            temporary=true}
             {self.canvas.update-object-for-undo
              {typeof x.obj} 'reshape x.obj commit=false
              self.translator
              local-unrotated=local-unrotated temporary=false}
             }
         }
       boxes}
  {if {not temporary}
      {self.canvas.commit-exchange}}
  } | method: reshape-objects-in-boxes


{define {update-dragging e:PointerEvent
                         snap-p:Point
                         current-select-boxes:list}:void
  x-offset:float  | unrotated, floats to help with casting
  y-offset:float  | unrotated, floats to help with casting
  width:float
  height:float

  {if {not self.just-selected}
      {self.reshape-objects-in-boxes current-select-boxes temporary=true}
      {set self.just-selected false}
      }

  {set x-offset y-offset {self.get-marker-offsets}}
  {cond {self.box.rotated
```

```
        {set width
             self.obj.internal-unrotated-width}
        {set height
             self.obj.internal-unrotated-height}
        }
       {else
        {set width
             self.obj.graphic-external-width}
        {set height
             self.obj.graphic-external-height}
        }} | else-cond
{cond {{!= 0 {bit-and self.marker-position
                      {bit-or CMARK1 CMARK4 CMARK7}}}
        {set self.scale-x {/ {+ {- x-offset}
                                width}
                             width}}
        }
       {{!= 0 {bit-and self.marker-position
                      {bit-or CMARK3 CMARK6 CMARK9}}}
        {set self.scale-x {/ {+ x-offset
                                width}
                             width}}
        }
       {else
        {set self.scale-x 1.0}
        }} | else-cond

{cond {{!= 0 {bit-and self.marker-position
                      {bit-or CMARK1 CMARK2 CMARK3}}}
        {set self.scale-y {/ {+ {- y-offset}
                                height}
                             height}}
        }
       {{!= 0 {bit-and self.marker-position
                      {bit-or CMARK7 CMARK8 CMARK9}}}
        {set self.scale-y {/ {+ y-offset
                                height}
                             height}}
        }
       {else
        {set self.scale-y 1.0}
        }} | else-cond

{self.reshape-objects-in-boxes current-select-boxes temporary=true}
} | method: update-dragging
```

```
  {define {done-dragging current-select-boxes:list}:void
    {self.reshape-objects-in-boxes current-select-boxes temporary=false}
    } | method: done-dragging
  } | class: ResizeBoxMarker



{define-class MoveBoxMarker {BoxMarker}

  offset:Point
  translator:{proc {Point}:Point}

  {define {init x-center:int=-1 y-center:int=-1 ...}
    | "rest" (...) is just a hack, so it looks kinda like the others
    {set self.translator
         {lambda {p:Point}:Point
           x:int={+ p.x self.offset.x}
           y:int={+ p.y self.offset.y}
           {return {new Point x y}}
           } | lambda
         } | set

    {cond {{or {= x-center -1} {= y-center -1}}
           {generic-apply-method BoxMarker 'init self
                                 {rest-as-list}
                                 x-center y-center CMARK5 width=0 height=0}

           }
          {else
           {generic-apply-method BoxMarker 'init self
                                 {rest-as-list} x-center y-center CMARK5}
           }
        }
    } | method: init

  {define {update-dragging e:PointerEvent
                           snap-p:Point
                           current-select-boxes:list}:void
    x-offset:int
    y-offset:int

    {if {not self.just-selected}
        {map {lambda {x}
               {x.obj.reshape self.translator temporary=true}
               }
             current-select-boxes}
        {set self.just-selected false}
```

```
      }
    {set x-offset y-offset {self.get-marker-offsets rotated=false}}
    {set self.offset {new Point x-offset y-offset}}
    {map {lambda {x}
           {x.obj.reshape self.translator temporary=true}
           }
         current-select-boxes}
    } | method: update-dragging

  {define {done-dragging current-select-boxes:list}:void
    {map {lambda {x}
           {self.canvas.update-object-for-undo
            {typeof x.obj} 'reshape x.obj commit=false
            self.translator local-unrotated=false}
           }
         current-select-boxes}
    {self.canvas.commit-exchange}
    } | method: done-dragging


  {define public {get-drag-image}:Graphic
    | ck: change the ruler marker to something that you like  Figure
    | out size..., which is the "outter limit of all moving things"
    {return {super.get-drag-image}}
    } | method: get-drag-image


  {define public {on-select}:void
    | ck: change the ruler marker to the 3-prong
    {super.on-select}
    } | method: on-select

  {define public {on-deselect}:void
    | ck: change the ruler marker back to its old self
    {super.on-deselect}
    } | method: on-deselect
  } | class: MoveBoxMarker


{define-class RotateBoxMarker {BoxMarker}
  marker-angle:float
  last-rotation-angle:float
  marker-position-from-center:Point

  {define {init x-center:int y-center:int marker-pos:int ...}

    | known types are: CMARK1, CMARK3, CMARK7, CMARK9
```

```
  {if {= 0 {bit-and marker-pos {bit-or CMARK1 CMARK3 CMARK7 CMARK9}}}
      {error {format void "~p~s~p"
                     "RotateBoxMarker: Bad type of marker \""
                     marker-pos
                     "\""}}
      } | if
  {generic-apply-method NewCanvasSelectMarker 'init self
                        {rest-as-list}
                        x-center y-center marker-pos
                        shape-style=FILLEDCIRCLE-MARKER-SHAPE}
  } | method: init


{define public {on-select}:void
  {super.on-select}
  {set self.marker-angle
       {arctangent2 {- {- self.y self.obj.object-center.y}}
                    {- self.x self.obj.object-center.x}}}
  {set self.marker-position-from-center
       {new Point {- self.x self.obj.object-center.x}
                  {- self.y self.obj.object-center.y}}}
  } | method: on-select


{define {update-dragging e:PointerEvent
                         snap-p:Point
                         current-select-boxes:list}:void
  mouse-x-offset:int
  mouse-y-offset:int
  new-angle:float

  {if {not self.just-selected}
      {map {lambda {x}
             {x.obj.rotate self.last-rotation-angle temporary=true}
             }
           current-select-boxes}
      {set self.just-selected false}
      }
  {set mouse-x-offset mouse-y-offset
       {self.get-marker-offsets rotated=false}}
  {set new-angle
       {arctangent2 {- {+ mouse-y-offset
                          self.marker-position-from-center.y}}
                    {+ mouse-x-offset
                       self.marker-position-from-center.x}}}
  {set self.last-rotation-angle {- new-angle self.marker-angle}}
  {map {lambda {x}
```

```
          {x.obj.rotate self.last-rotation-angle temporary=true}
          }
        current-select-boxes}
    } | method: update-dragging

  {define {done-dragging current-select-boxes:list}:void
    {map {lambda {x}
          {self.canvas.update-object-for-undo
           {typeof x.obj} 'rotate x.obj commit=false
           self.last-rotation-angle}
          }
        current-select-boxes}
    {self.canvas.commit-exchange}
    } | method: done-dragging
  } | class: RotateBoxMarker
```

A.3.5 rect.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains the event handler and the CanvasObject for
| representing rectangles.

{define-class RectangleModeEventHandler {NewCanvasEventHandler}
  from-center:bool
  first-point:Point
  last-pos:Point
  creating:bool

  {define {init}
    {set self.first-point {new Point 0 0}}
    {set self.last-pos {new Point 0 0}}
    } | method: init

  {define {toggle-creating}:void
    {set self.creating {not self.creating}}
    {self.new-descriptions}
    {if self.creating
        {self.canvas.lock-semaphore 'mode}
        {self.canvas.release-semaphore 'mode}
        }
    } | method: toggle-creating

  {define {new-descriptions}:void
    {if self.creating
        {self.canvas.set-event-descriptions
```

```
      pointer-press="Place Corner"
      shiftify="Square"
      }
     {self.canvas.set-event-descriptions
      pointer-press="Start Rectangle"
      }
     }
  } | method: new-descriptions


{define {pointer-press e:PointerEvent snap-point:Point}:void
  corner:Point
  obj:CanvasRectangleObject

  {cond {self.creating
         | need to wrap things up
         {if self.from-center
             {set corner
                  {centerandcorner->diag self.first-point snap-point}}
             {set corner self.first-point}
             }
         {if {neq? {self.canvas.get-canvas-property 'fill-color} void}
             {set obj
                  {new CanvasRectangleObject corner snap-point true}}
             {set obj
                  {new CanvasRectangleObject corner snap-point false}}
             }
         {obj.add-to-canvas self.canvas.canvas.Sheet
                            copy-properties=true commit=true}
         {self.toggle-creating}
         }
        {else
         | need to start making a rectangle
         {set self.first-point.x snap-point.x}
         {set self.first-point.y snap-point.y}
         {set self.last-pos.x snap-point.x}
         {set self.last-pos.y snap-point.y}
         {set self.from-center {not {self.canvas.get-application-property
                                     'draw-from-corner}}}
         | this should just make a point...
         {rubberband-rectangle self.canvas.canvas.Sheet
                               self.first-point.x self.first-point.y
                               self.first-point.x self.first-point.y}
         {self.toggle-creating}
         }
        } | cond
  } | method: pointer-press
```

```
{define {draw-rect press1:Point
                   press2:Point
                   ignore-invalidate:bool=true}:void
  corner:Point

  {if {not self.creating}
      {return}}
  {cond {{and {not ignore-invalidate} self.invalidated}
         {set self.invalidated false}
         }
        {self.from-center
         {set corner {centerandcorner->diag press1 press2}}
         {rubberband-rectangle self.canvas.canvas.Sheet
                               corner.x corner.y press2.x press2.y}
         }
        {else
         {rubberband-rectangle self.canvas.canvas.Sheet
                               press1.x press1.y
                               press2.x press2.y}
         }
        } | cond
  } | method: draw-rect


{define {pointer-motion e:PointerEvent snap-point:Point}:void
  corner:Point

  {if {not self.creating}
      {return}}
  | first, erase the old rectangle
  {self.draw-rect self.first-point self.last-pos ignore-invalidate=false}
  | now draw the new rectangle
  {self.draw-rect self.first-point snap-point}
  | oh, and don't forget to update that last-pos... teehee
  {set self.last-pos.x snap-point.x}
  {set self.last-pos.y snap-point.y}
  } | method: pointer-motion


{define {key-press e:KeyEvent}:void
  {cond {{not self.creating}
         {return}
         }
        {{or {= e.key 127} {= e.key 8}}  | backspace/delete
         | stop creating
         {self.draw-rect self.first-point self.last-pos
```

```
                          ignore-invalidate=false}
          {self.toggle-creating}
          }
         {else
          {return}
          }} | else-cond
    } | method: key-press

  {define {cancel}:void
    e:CanvasKeyEvent
    {when self.creating
      {set e {new CanvasKeyEvent}}
      {set e.key 127}
      {self.key-press e}
      } | when
    } | method: cancel

  {define {draw gc:GraphicContext}:void
    {self.draw-rect self.first-point self.last-pos ignore-invalidate=false}
    } | method: draw

  {define {selected}:void
    {self.canvas.set-message t="Rectangle mode selected"}
    {self.new-descriptions}
    } | method: selected

  {define {popup-dialog-box}:void
    | ck: actually do this sometime
    {super.popup-dialog-box}
    } | method: popup-dialog-box

  } | class: RectangleEventHandler



{define-class CanvasRectangleObject {NewCanvasDrawingObject}
  upper-left:Point  | canvas, rotated point
  lower-right:Point | canvas, rotated point
  upper-right:Point
  lower-left:Point
  filled:bool

  {define {init corner1:Point corner2:Point filled:bool}
    {set self.upper-left self.lower-right
         {twocorners->origdiag corner1 corner2}}
    {set self.filled filled}
    {set self.upper-right {new Point self.lower-right.x self.upper-left.y}}
```

```
  {set self.lower-left {new Point self.upper-left.x self.lower-right.y}}
  {super.init self.upper-left.x self.upper-right.y
              {- self.lower-right.x self.upper-left.x}
              {- self.lower-right.y self.upper-left.y}}
  {set self.marker-type
       {bit-or self.marker-type RESIZEABLE FREEROTATEABLE}}
  {self.canvas-object-properties.set-one-property 'line-width void}
  {self.canvas-object-properties.set-one-property 'line-color void}
  {self.canvas-object-properties.set-one-property 'fill-color void}
  } | method: init

{define public {re-initialize}
  minx:int
  miny:int
  maxx:int
  maxy:int
  up-left:Point
  low-right:Point

  | first, do the graphic resize thing
  {set minx maxx miny maxy {points-minmax
                            {make-list self.upper-left
                                       self.upper-right
                                       self.lower-right
                                       self.lower-left}}}
  {self.graphic-resize minx maxx miny maxy}
  | next, make sure the points are labeled correctly
  {set up-left
       {rotate-point self.upper-left rad-angle={- self.rotation-angle}
                     center=self.object-center}}
  {set low-right
       {rotate-point self.lower-right rad-angle={- self.rotation-angle}
                     center=self.object-center}}
  {set up-left low-right {twocorners->origdiag up-left low-right}}
  {set self.upper-left
       {rotate-point up-left rad-angle=self.rotation-angle
                     center=self.object-center}}
  {set self.lower-right
       {rotate-point low-right rad-angle=self.rotation-angle
                     center=self.object-center}}
  | do the rotated resize thing
  {self.set-internal-corners self.upper-left self.lower-right}
  | now, reset the other two corners, in case they changed
  {set self.upper-right.x {round {+ self.upper-left.x
                                    {* self.internal-unrotated-width
                                       {cosine self.rotation-angle}}}}}
  {set self.upper-right.y {round {- self.upper-left.y
```

```
                                        {* self.internal-unrotated-width
                                           {sine self.rotation-angle}}}}}
  {set self.lower-left.x
       {round {+ self.upper-left.x
                 {* self.internal-unrotated-height
                    {cosine {- self.rotation-angle {/ PI 2}}}}}}}}
  {set self.lower-left.y
       {round {- self.upper-left.y
                 {* self.internal-unrotated-height
                    {sine {- self.rotation-angle {/ PI 2}}}}}}}}
  } | re-initialize


{define public {draw gc:GraphicContext}:void
  upper-left:Point={self.canvas-point-to-local-coord self.upper-left}
  upper-right:Point={self.canvas-point-to-local-coord self.upper-right}
  lower-left:Point={self.canvas-point-to-local-coord self.lower-left}
  lower-right:Point={self.canvas-point-to-local-coord self.lower-right}
  line-color:any={self.get-object-property 'line-color}
  line-width:int={self.get-object-property 'line-width}


  {cond {{or {not self.filled} {!= 0 self.rotation-angle}}
         | start with the inside
         {if self.filled
             {gc.fill-polygon {self.get-object-property 'fill-color}
                              upper-left.x upper-left.y
                              upper-right.x upper-right.y
                              lower-right.x lower-right.y
                              lower-left.x lower-left.y}
             }
         | now, need to draw each line separately
         {gc.draw-line upper-left.x upper-left.y
                       upper-right.x upper-right.y
                       line-color line-width}
         {gc.draw-line upper-right.x upper-right.y
                       lower-right.x lower-right.y
                       line-color line-width}
         {gc.draw-line lower-right.x lower-right.y
                       lower-left.x lower-left.y
                       line-color line-width}
         {gc.draw-line lower-left.x lower-left.y
                       upper-left.x upper-left.y
                       line-color line-width}
         }
        {else
         {gc.draw-rect upper-left.x upper-left.y
```

```
                    lower-right.x lower-right.y
                    {self.get-object-property 'fill-color}
                    line-color line-width}
        }
       } | cond
 } | method: draw

{define {reshape translator:{proc {Point}:Point}
                p:Point=void
                local-unrotated:bool=false
                temporary:bool=false}:void
  upper-left:Point
  upper-right:Point
  lower-right:Point
  lower-left:Point

  {if {not {void? p}}
      {throw {new Exception
                  {format void "~p~p~p"
                               "Not allowed to "
                               "move a point of a rectangle.  "
                               "Try changing to a polygon."}}}}

  {cond
   {{not local-unrotated}
    {set upper-left self.upper-left}
    {set upper-right self.upper-right}
    {set lower-left self.lower-left}
    {set lower-right self.lower-right}
    }
   {else
    {set upper-left {self.canvas-point-to-local-coord
                     self.upper-left unrotate=true new-origin=true}}
    {set upper-right {self.canvas-point-to-local-coord
                      self.upper-right unrotate=true new-origin=true}}
    {set lower-left {self.canvas-point-to-local-coord
                     self.lower-left unrotate=true new-origin=true}}
    {set lower-right {self.canvas-point-to-local-coord
                      self.lower-right unrotate=true new-origin=true}}
    }} | else-cond
  {set upper-left {translator upper-left}}
  {set upper-right {translator upper-right}}
  {set lower-left {translator lower-left}}
  {set lower-right {translator lower-right}}

  {when local-unrotated
    | need adjust back
```

```
    {set upper-left {self.local-point-to-canvas-coord
                     upper-left rotate=true new-origin=true}}
    {set upper-right {self.local-point-to-canvas-coord
                      upper-right rotate=true new-origin=true}}
    {set lower-left {self.local-point-to-canvas-coord
                     lower-left rotate=true new-origin=true}}
    {set lower-right {self.local-point-to-canvas-coord
                      lower-right rotate=true new-origin=true}}
    } | when

  {cond {temporary
         {rubberband-lines
          self.canvas {make-list upper-left upper-right
                                 lower-right lower-left upper-left}}
         }
        {else
         {set self.upper-left upper-left}
         {set self.lower-right lower-right}
         {set self.upper-right upper-right}
         {set self.lower-left lower-left}
         {self.re-initialize}
         }
        } | cond
  } | method: reshape

{define {rotate radian-angle:float temporary:bool=false}:void
  upper-left:Point
  upper-right:Point
  lower-right:Point
  lower-left:Point

  {set upper-left {rotate-point self.upper-left rad-angle=radian-angle
                                center=self.object-center}}
  {set upper-right {rotate-point self.upper-right rad-angle=radian-angle
                                 center=self.object-center}}
  {set lower-left {rotate-point self.lower-left rad-angle=radian-angle
                                center=self.object-center}}
  {set lower-right {rotate-point self.lower-right rad-angle=radian-angle
                                 center=self.object-center}}
  {cond {temporary
         {rubberband-lines self.canvas
                           {make-list upper-left upper-right
                                      lower-right lower-left
                                      upper-left}}
         }
        {else
         {set self.upper-left upper-left}
```

```
        {set self.upper-right upper-right}
        {set self.lower-left lower-left}
        {set self.lower-right lower-right}
        {set self.rotation-angle {+ self.rotation-angle radian-angle}}
        {self.re-initialize}
        }
       } | cond
  } | method: rotate

{define {distance-to-object p:Point}:float
  dist:float
  {set dist {point-to-lineseg-distance p self.upper-left
                                       self.upper-right}}
  {set dist {min dist {point-to-lineseg-distance p self.upper-right
                                                 self.lower-right}}}
  {set dist {min dist {point-to-lineseg-distance p self.lower-right
                                                 self.lower-left}}}
  {set dist {min dist {point-to-lineseg-distance p self.lower-left
                                                 self.upper-left}}}
  {return dist}
  } | method: distance-to-object

{define {copy-guts obj:NewCanvasObject}:void
  obj-local:CanvasRectangleObject=obj   | for type checking
  | points are rotated by the super.
  {set obj-local.upper-left {rotate-point self.upper-left
                                     rad-angle={- self.rotation-angle}
                                     center=self.object-center}}
  {set obj-local.upper-right {rotate-point self.upper-right
                                      rad-angle={- self.rotation-angle}
                                      center=self.object-center}}
  {set obj-local.lower-right {rotate-point self.lower-right
                                      rad-angle={- self.rotation-angle}
                                      center=self.object-center}}
  {set obj-local.lower-left {rotate-point self.lower-left
                                     rad-angle={- self.rotation-angle}
                                     center=self.object-center}}
  {set obj-local.filled self.filled}
  {super.copy-guts obj}
  } | method: copy-guts


{define {save-to-file fout:OutputPort}:void
  | stored as (points in Canvas coordinates, but unrotated):
  | <{type->symbol {typeof self}}>
  | <upper-left.x> <upper-left.y>
  | <lower-right.x> <lower-right.y>
```

```
  | <filled>
  | <<<super.save-to-file>>>

  {format fout "~p~%" {type->symbol {typeof self}}}
  {format fout "~d ~d~%" self.upper-left.x self.upper-left.y}
  {format fout "~d ~d~%" self.lower-right.x self.lower-right.y}
  {format fout "~d~%"    self.filled}
  {super.save-to-file fout}
  } | method: save-to-file

{define {read-from-file fin:InputPort}:void
  input-list:list
  temp-text:text
  x:int y:int
  upper-left:Point
  lower-right:Point
  filled:bool

  {set input-list {format-in fin "~s~d~d~d~d~d"}}
  {if {!= {list-length input-list} 6}
      {error {format void "Improper input file, bad ~s"
                     {type->symbol {typeof self}}}}}
  | get the items from the list
  | item 1
  {set temp-text {car input-list}}
  {set input-list {cdr input-list}}
  {if {not {eq? {text->symbol temp-text} {type->symbol {typeof self}}}}
      {error {format void "Improper input file, bad ~s"
                     {type->symbol {typeof self}}}}
      }
  | item 2 and 3
  {set x {car input-list}}
  {set input-list {cdr input-list}}
  {set y {car input-list}}
  {set input-list {cdr input-list}}
  {set upper-left {new Point x y}}
  | items 4 and 5
  {set x {car input-list}}
  {set input-list {cdr input-list}}
  {set y {car input-list}}
  {set input-list {cdr input-list}}
  {set lower-right {new Point x y}}
  | items 6
  {set filled {car input-list}}
  {set input-list {cdr input-list}}
  | imitate init
  {set self.upper-left upper-left}
```

```
  {set self.lower-right lower-right}
  {set self.filled filled}
  {set self.upper-right {new Point lower-right.x upper-left.y}}
  {set self.lower-left {new Point upper-left.x lower-right.y}}
  {super.read-from-file fin}
  } | method: read-from-file


{define {export-draw fout:OutputPort}:void
  {if {eq? {type->symbol {typeof self}} {type->symbol CanvasLineObject}}
      {format fout "    | CanvasRectangleObject stuff~%"}
      }
  {format fout "    {let~%"}
  {format fout "      line-color:any=~p~%"
          {color->text {self.get-object-property 'line-color}
                       quote-sym=true}}
  {format fout "      line-width:int=~d~%"
          {self.get-object-property 'line-width}}
  {cond {{or {not self.filled} {!= self.rotation-angle 0}}
         {format fout "      | rotated rectangle~%"}
         {when self.filled
           {format fout "      | Start with the filling~%"}
           {format fout "~p~p ~d ~d ~d ~d ~d ~d ~d}~%"
                   "      {gc.fill-polygon "
                   {color->text {self.get-object-property 'fill-color}
                                quote-sym=true}
                   self.upper-left.x self.upper-left.y
                   self.upper-right.x self.upper-right.y
                   self.lower-right.x self.lower-right.y
                   self.lower-left.x self.lower-left.y}
           } | when
         {format fout "~p~p~%"
                 "      | and now the border, "
                 "which may overwrite filling"}
         {format fout "~p~d ~d ~d ~d~p~%"
                 "      {gc.draw-line "
                 self.upper-left.x self.upper-left.y
                 self.upper-right.x self.upper-right.y
                 " line-color line-width}"}
         {format fout "~p~d ~d ~d ~d~p~%"
                 "      {gc.draw-line "
                 self.upper-right.x self.upper-right.y
                 self.lower-right.x self.lower-right.y
                 " line-color line-width}"}
         {format fout "~p~d ~d ~d ~d~p~%"
                 "      {gc.draw-line "
                 self.lower-right.x self.lower-right.y
```

```
                self.lower-left.x self.lower-left.y
                " line-color line-width}"}
         {format fout "~p~d ~d ~d ~d~p~%"
                "      {gc.draw-line "
                self.lower-left.x self.lower-left.y
                self.upper-left.x self.upper-left.y
                " line-color line-width}"}
         } | rotated
        {else
         {format fout "~p~d ~d ~d ~d ~p~p~%"
                "      {gc.draw-rect "
                self.upper-left.x self.upper-left.y
                self.lower-right.x self.lower-right.y
                {color->text {self.get-object-property 'fill-color}
                             quote-sym=true}
                " line-color line-width}"
                }
         }} | else-cond
    {format fout "    } | let~%"}
    {super.export-draw fout}
    {format fout "~%"}
    } | method: export-draw
  } | class: NewRectangleObject
```

A.4 Visual Components

A.4.1 canvas.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This defines all strictly canvas stuff.  That would be the
| CanvasSheet, the CanvasApplication, and CanvasAndRulers.

{define-constant canvas-action-type:type={proc {NewCanvasSheet}:void}}
{define {no-canvas-action x:NewCanvasSheet}:void {return}}
{define-macro public {canvas-action l:list}
  {return `{lambda {void:NewCanvasSheet}:void ,@l}}}

{define-mixin-class CanvasAndRulers {Table}
  Sheet:NewCanvasSheet
  Container:NewCanvasApplication
  sheetcol:int
  sheetrow:int

  {define {init sheet:NewCanvasSheet
```

```
                sheetcol:int=0 sheetrow:int=0}
    {super.init cell-border-width=1 cell-border-style='black}
    {set self.Sheet sheet}
    {self.add-object self.Sheet sheetrow sheetcol}
    {set self.sheetcol sheetcol}
    {set self.sheetrow sheetrow}
    } | method: init

  {define {set-container app:NewCanvasApplication}:void
    {set self.Container app}
    } | method: set-container

  {define {show-rulers}:void
    {error "show-rulers not overwritten"}
    } | method: show-rulers
  {define {hide-rulers}:void
    {error "hide-rulers not overwritten"}
    } | method: hide-rulers

  {define {change-sheet new-canvas:NewCanvasSheet}:NewCanvasSheet
    old-canvas:NewCanvasSheet=self.Sheet
    {set self.Sheet new-canvas}
    {self.add-object new-canvas self.sheetrow self.sheetcol}
    {new-canvas.set-container self}
    {old-canvas.copy-state new-canvas}
    {return old-canvas}
    } | method: change-sheet
  } | class: CanvasAndRulers


{define-class 2DCanvasAndRulers {CanvasAndRulers}
  Hruler:NewStraightRuler    | may be void, but only if Vruler is void
  Vruler:NewStraightRuler    | may be void, but only if Hruler is void
  sheet-width:int
  sheet-height:int
  ruler-height:int
  rulers-showing:bool
  vert-ruler-on-right:bool

  {define {init rulers?:bool=true
                vert-ruler-on-right:bool=true
                width:int=CANVAS-SHEET-AND-RULERS-WIDTH
                height:int=CANVAS-SHEET-AND-RULERS-HEIGHT
                ruler-height:int=RULER-HEIGHT}
    sheet:NewCanvasSheet

    {set self.sheet-width {- width ruler-height}}
```

```
  {set self.sheet-height {- height ruler-height}}
  {set sheet {new NewCanvasSheet
                 width=self.sheet-width
                 height=self.sheet-height}}
  {super.init sheet sheetcol=1 sheetrow=1}
  {self.Sheet.set-container self}
  {set self.ruler-height ruler-height}
  {set self.vert-ruler-on-right vert-ruler-on-right}

  {set self.Hruler {new NewStraightRuler
                       width={- width ruler-height}}}
  {if vert-ruler-on-right
      {set self.Vruler {new NewStraightRuler
                           horizontal=false
                           width={- height ruler-height}
                           on-top-left=false}}
      {set self.Vruler {new NewStraightRuler
                           horizontal=false
                           width={- height ruler-height}}}
      }

  {when rulers?
    {self.show-rulers}
    {set self.rulers-showing rulers?}
    } | when
  } | method: init


{define {show-rulers}:void
  {when {not self.rulers-showing}
    | only do something if the rulers are currently hiding
    {self.Sheet.set-property width=self.sheet-height
                             height=self.sheet-width}
    {self.add-object self.Hruler 0 1}
    {set self.Hruler.showing true}
    {if self.vert-ruler-on-right
        {self.add-object self.Vruler 1 2}
        {self.add-object self.Vruler 1 0}
        }
    {set self.Vruler.showing true}
    {set self.rulers-showing true}
    }
  } | method: show-rulers

{define {hide-rulers}:void
  {when self.rulers-showing
    | only do something if the rulers are currently showing
```

```
      {self.Sheet.set-property
       width={+ self.sheet-width self.ruler-height}
       height={+ self.sheet-height self.ruler-height}}
      {self.remove-object self.Hruler}
      {set self.Hruler.showing false}
      {self.remove-object self.Vruler}
      {set self.Vruler.showing false}
      {set self.rulers-showing false}
      }
    } | method: hide-rulers

  {define {new-sheet}:NewCanvasSheet
    sheet:NewCanvasSheet
    {if self.rulers-showing
        {set sheet {new NewCanvasSheet width=self.sheet-width
                                       height=self.sheet-height}}
        {set sheet {new NewCanvasSheet
                        width={+ self.sheet-width self.ruler-height}
                        height={+ self.sheet-height self.ruler-height}}}
        }
    {sheet.set-container self}
    {return {self.change-sheet sheet}}
    } | method: new-sheet
  } | class: 2DCanvasAndRulers



{define-class NewCanvasSheet {Frame EventHandler}
  MEH:NewCanvasEventHandler     | Mode EventHandler
  Objects:list    | of NewCanvasObject
  hidden-objects:list
  select-boxes:list | of CanvasBox
  Container:2DCanvasAndRulers
  grid:Grid
  canvas-properties:CanvasPropertyList
  undo-buffer:UndoBuffer
  last-object-drawn:NewCanvasObject

  {define {init width:int=CANVAS-SHEET-WIDTH
                height:int=CANVAS-SHEET-HEIGHT}
    {invoke-method Frame 'init self
      valign='top
      width=width height=height
      border-width=1 border-style='black}
    | bug: can't access init without "super.init"
    | {invoke-method EventHandler 'init self}
    {set self.canvas-properties {new CanvasPropertyList}}
```

```
  {set self.undo-buffer {new UndoBuffer 10}}
  {set self.grid {new Grid width height}}
  } | method: init


{define {copy-state to-canvas:NewCanvasSheet}:void
  {map
   {lambda {x}
     {to-canvas.canvas-properties.set-one-property
          x {self.get-canvas-property x}}
     }
   {self.canvas-properties.get-all-property-names}
   } | map
  {to-canvas.change-MEH self.MEH}
  } | method: copy-state

{define {set-container canvas-and-rulers:2DCanvasAndRulers}:void
  {set self.Container canvas-and-rulers}
  {self.grid.set-canvas canvas-and-rulers.Container}
  } | method: set-container

{define {change-MEH newmode:NewCanvasEventHandler}:void
  selected-objects:list

  {if {self.check-semaphore 'mode}
      | can't be changed
      {error {format void "~p~p"
                     "Call to NewCanvasSheet.change-MEH while "
                     "mode-semaphore is on."}}
      }
  {if {eq? newmode self.MEH}
      {return}}
  {if {isa NewCanvasEditMode newmode}
      {cond {{void? self.last-object-drawn}
             {set selected-objects {map {lambda {x}
                                          {return x.obj}}
                                        self.select-boxes}}
             }
            {else
             {set selected-objects
                  {new list self.last-object-drawn void}}
             {set self.last-object-drawn void}
             }}
      }
  {if {not {void? self.MEH}}
      {self.MEH.cancel}}
  {set self.MEH newmode}
```

```
  {self.set-event-descriptions}
  {self.message ""}
  {newmode.selected}
  {cond {{isa NewCanvasDrawMode newmode}
         {set self.last-object-drawn void}
         }
        {{isa NewCanvasEditMode newmode}
         {{cast NewCanvasEditMode newmode}.set-selected-objects
                                                  selected-objects}
         }
        } | cond
  | so, if they changed it, that means they probably clicked
  | elsewhere...
  {request-key-focus self}
  } | method: change-MEH


{define public {add-select-box obj:CanvasBox
                               unselect-previous:bool=true}:Graphic
  | first, deal with unselecting old stuff
  {if {and unselect-previous {not {void? self.select-boxes}}}
      {map {lambda {x}    | where x is a CanvasBox
             {x.remove-from-canvas}
             }
           self.select-boxes}}
  | now select the new
  {when {not {void? obj}}
    {set self.select-boxes {cons obj self.select-boxes}}
    | put the select box before all the objects, so it's visible.
    | But not on top.  (Markers go on top.)
    {set obj {self.insert-object obj after={car self.Objects}}}
    {cond {{void? obj.canvas}
           {obj.set-canvas self}}
          {{neq? obj.canvas self}
           {error "Adding select box to canvas other than its own."}
           }
          | else fine
          } | cond
    } | when
  {return obj}
  } | method: add-select-box


{define {remove-select-box obj:CanvasBox}:void
  {set self.select-boxes {delq! obj self.select-boxes}}
  {self.remove-object obj}
  } | method: remove-select-box
```

```
{define {add-canvas-object obj:NewCanvasObject}:void
  depth:int
  templist:list
  templist2:list

  | start with the obvious
  {set self.last-object-drawn obj}

  | need to find the first object with the same depth (or lower
  | depth) as obj.  So it can be on top, but not too far on top.
  {set depth {obj.get-object-property 'depth}}
  {set templist self.Objects}
  {while {and {not {void? templist}}
              {not {void? {cdr templist}}}
              {< {{cadr templist}.get-object-property 'depth} depth}}
    {set templist {cdr templist}}
    }

  {cond {{void? templist}
         | there were never any objects in the first place
         {set self.Objects {cons obj self.Objects}}
         {self.append-object obj}
         }
        {{>= {{car templist}.get-object-property 'depth} depth}
         | This means that the new guy was supposed to be on top
         {set self.Objects {cons obj self.Objects}}
         {self.insert-object obj after={cadr self.Objects}}
         }
        {else
        | Note that "else" covers both the case where the cdr if
        | void, and newguy is after everything, and the case where
        | I'm just putting the new guy on the bottom.
         {set templist.rest {cons obj templist.rest}}
         {self.insert-object obj before=templist.first}
         }
        } | cond
  | look at prefs and see if you need to change to the default mode
  {when {not {self.get-application-property 'mode-centric}}
    {let application:NewCanvasApplication=self.Container.Container
         toolbar:NewModesToolbar={application.modebar.get-current-value}
      {cond {{or {void? toolbar}
                 {not {isa NewRadioToolbar toolbar}}}
             {return}
             }
            {{not {void? toolbar.default-button}}
```

```
            {toolbar.default-button.pointer-press
              {new CanvasPointerEvent}}
            }
           {else
            {toolbar.set-active void}
            }
           } | cond
      } | let
    } | when
  } | method: add-canvas-object


{define {remove-canvas-object obj:NewCanvasObject}:void
  box:CanvasBox
  {if {void? {memq obj self.Objects}}
      {error {format void "~p~p"
                     "Removing an object from NewCanvasSheet not "
                     "previously added."}}
      }

  {set self.Objects {delq! obj self.Objects}}
  {if {not {void? {memq obj self.hidden-objects}}}
      {set self.hidden-objects {delq! obj self.hidden-objects}}
      {super.remove-object obj}
      }
  | and remove any select boxes
  {set box {self.get-object-CanvasBox obj}}
  {if {not {void? box}}
      {box.remove-from-canvas}
      }
  } | method: remove-canvas-object


{define public {objects-within-tolerance p:Point
                                         tolerance:int=-1
                                         visible-only:bool=true
                                         }:{return list list}
  return-objects:list    | of NewCanvasObject
  return-distances:list  | of floats
  objects-to-check:list=self.Objects

  {if {< tolerance 0}
      {set tolerance {self.get-application-property 'select-tolerance}}}
  {while {not {void? objects-to-check}}
    {let curobj:NewCanvasObject
         curdist:float
      {if {and visible-only
```

```
              {not {void? {memq {car objects-to-check}
                                self.hidden-objects}}}}
          {continue}}
      {set curobj {car objects-to-check}}
      {set curdist {curobj.distance-to-object p}}
      {set objects-to-check {cdr objects-to-check}}
      {when {< curdist tolerance}
        {set return-objects {cons curobj return-objects}}
        {set return-distances {cons curdist return-distances}}
        } | when
      } | let
    } | while
  {return return-objects return-distances}
  } | method: objects-within-tolerance


{define public {closest-object p:Point
                               tolerance:int=-1
                               visible-only:bool=true}:NewCanvasObject
  lobj:list
  ldist:list
  mindist:float=MAXINT
  minobj:NewCanvasObject

  {if {< tolerance 0}
      {set tolerance {self.get-application-property 'select-tolerance}}}
  {set lobj ldist
       {self.objects-within-tolerance p
                                      tolerance=tolerance
                                      visible-only=visible-only}}
  {while {not {void? ldist}}
    {let curobj:NewCanvasObject={car lobj}
         curdist:float={car ldist}
      {set lobj {cdr lobj}}
      {set ldist {cdr ldist}}
      {when {< curdist mindist}
        {set minobj curobj}
        {set mindist curdist}
        }
      } | let
    } | while
  {return minobj}  | which may be void, if the lists were void
  } | method: closest-object


{define {exchange-single-canvas-object
         cur-obj:NewCanvasObject
```

```
       new-obj:NewCanvasObject
       commit:bool=false
       undo-able:bool=true}:void
sub-Objects:list
box:CanvasBox

{cond {{and {void? cur-obj} {void? new-obj}}
       {return}
       }
      {{void? cur-obj}
       | just add the object
       {self.add-canvas-object new-obj}
       {new-obj.added-to-canvas self}
       }
      {{void? new-obj}
       | just remove the old guy
       {self.remove-canvas-object cur-obj}
       }
      {else
       | get it changed in internal Objects
       {set sub-Objects {memq cur-obj self.Objects}}
       {if {void? sub-Objects}
           {error {format void "~p~p"
                          "Trying to exchange an object in "
                          "NewCanvasSheet not previously added."}}
           }
       {set sub-Objects.first new-obj}
       | do the same with hidden
       {set sub-Objects {memq cur-obj self.hidden-objects}}
       {if {not {void? sub-Objects}}
           {set sub-Objects.first new-obj}}
       | and finally, switch it in a select box
       {set box {self.get-object-CanvasBox cur-obj}}
       {if {not {void? box}}
         {box.change-object new-obj}}
       | finally, the graphical stuff
       {self.replace-object cur-obj new-obj}
       {new-obj.added-to-canvas self}
       } | else
      } | cond
{when undo-able
  {self.undo-buffer.add-item {make-list cur-obj new-obj} commit=commit}
  }
| hack: and just for personal preference.
{self.message ""}
} | method: exchange-single-canvas-object
```

```
{define {exchange-canvas-objects current-objs:any
                                 new-objs:any
                                 commit:bool=false}:void
  isa-list-cur:bool={isa list current-objs}
  isa-list-new:bool={isa list new-objs}

  {cond {{and {isa list current-objs} {isa list new-objs}}
         | void is not a list, so we have two lists with items
         {if {!= {list-length current-objs} {list-length new-objs}}
             {error "Exchanging lists of objects of unequal length."}
             }
         {dotimes {i {list-length current-objs}}
           {self.exchange-single-canvas-object
            {car current-objs} {car new-objs} commit=false}
           {set current-objs {cdr current-objs}}
           {set new-objs {cdr new-objs}}
           } | dotimes
         {if commit {self.commit-exchange}}
         } | cond-two-lists
        {{and {not {isa list current-objs}} {not {isa list new-objs}}}
         | two items, both possibly void
         {self.exchange-single-canvas-object current-objs
                                             new-objs
                                             commit=commit}
         } | cond: items and void
        {{and {isa list current-objs} {void? new-objs}}
         {dotimes {i {list-length current-objs}}
           {self.exchange-single-canvas-object
            {car current-objs} void commit=false}
           {set current-objs {cdr current-objs}}
           }
         {if commit {self.commit-exchange}}
         } | cond: list-current and void-new
        {{and {void? current-objs} {isa list new-objs}}
         {dotimes {i {list-length current-objs}}
           {self.exchange-single-canvas-object
            void {car new-objs} commit=false}
           {set new-objs {cdr new-objs}}
           }
         {if commit {self.commit-exchange}}
         } | cond: void-current and list-new
        {else
         {error "Exchanging a mixture of lists and items"}
         }
        } | cond
  } | method: exchange-canvas-objects
```

```
{define {commit-exchange}:void
  {self.undo-buffer.commit}
  } | method: commit-exchange

{define {update-object-for-undo object-type:type
                                 method:symbol
                                 object:NewCanvasObject
                                 commit:bool=true
                                 ...}:void
  old-obj:NewCanvasObject
  new-obj:NewCanvasObject

  {if {isa CanvasBox object}
      {set old-obj {cast CanvasBox object}.obj}
      {set old-obj object}}
  {set new-obj {invoke-method object-type 'copy object}}
  {myapply-method {typeof new-obj} method new-obj {rest-as-list}}
  {self.exchange-single-canvas-object old-obj new-obj commit=commit}
  } | method: update-object-for-undo

{define {undo}:void
  | Items are stored as "{make-list old new}"
  multi-list:list
  single-list:list
  {set multi-list {self.undo-buffer.get-undo-items}}
  {if {void? multi-list}
      {self.error "Nothing to undo."}
      }
  {dotimes {i {list-length multi-list}}
    {set single-list {car multi-list}}
    {self.exchange-single-canvas-object {cadr single-list}
                                        {car single-list}
                                        undo-able=false}
    {set multi-list {cdr multi-list}}
    } | dotimes
  } | method: undo

{define {redo}:void
  | Items are stored as "{make-list old new}"
  multi-list:list
  single-list:list
  {set multi-list {self.undo-buffer.get-redo-items}}
  {if {void? multi-list}
      {self.error "Can't redo: Last action wasn't an undo"}
      }
  {dotimes {i {list-length multi-list}}
```

```
    {set single-list {car multi-list}}
    {self.exchange-single-canvas-object {car single-list}
                                        {cadr single-list}
                                    undo-able=false}
    {set multi-list {cdr multi-list}}
    } | dotimes
  } | method: redo


{define public {resize width:int=-1 height:int=-1}:void
  {if {!= width -1}
      {self.set-property width=width}}
  {if {!= height -1}
      {self.set-property height=height}}
  {self.grid.resize width=width height=height}
  {self.invalidate}
  } | method: resize

{define {get-object-CanvasBox obj:NewCanvasObject}:CanvasBox
  temp-list:list=self.select-boxes
  {while {not {void? temp-list}}
    {if {eq? obj {car temp-list}.obj}
        {return {car temp-list}}
        {set temp-list {cdr temp-list}}
        }
    }
  {return void}
  } | method: get-object-CanvasBox

{define {hide-canvas-object obj:NewCanvasObject}:void
  temp-list:list
  temp-box:CanvasBox=void

  {if {not {void? {memq obj self.hidden-objects}}}
      | it's already hidden
      {return}}

  {if {void? {memq obj self.Objects}}
      | it's not an object we know about
      {throw {new Exception "Trying to hide an object not added."}}}

  {self.remove-object obj}
  {set self.hidden-objects {cons obj self.hidden-objects}}

  {set temp-box {self.get-object-CanvasBox obj}}
  {if {not {void? temp-box}}
      {temp-box.remove-from-canvas}
```

```
    }
  } | method: hide-canvas-object


{define {show-canvas-object obj:NewCanvasObject}:void
  temp-list:list

  {if {void? {memq obj self.Objects}}
      | it has been removed while it was hidden
      {throw {new Exception "Trying to show unknown object"}}
      }

  | ok, a little clean-up
  {set self.hidden-objects {delq! obj self.hidden-objects}}

  | special stuff to be done if it's the first object
  {when {eq? {car self.Objects} obj}
    | Ok, obj needs to go on top, but under the select boxes...
    {if {not {void? {cdr self.Objects}}}
        | just put it on top of the second guy
        {self.insert-object obj after={cadr self.Objects}}
        | there can't be any select boxes, you can just append it
        {self.append-object obj}
        }
    {return}
    }

  | we need to find what object it goes under
  {set temp-list self.Objects}
  {while {not {eq? {cadr temp-list} obj}}
    {set temp-list {cdr temp-list}}
    }
  | so, the first object here is what's above our guy
  {self.insert-object obj before={car temp-list}}
  } | method: show-canvas-object


{define {hide-layers below-current-layer:bool=false
                     above-current-layer:bool=true
                     current-layer:bool=true}:void

  layers-showing:int={self.get-application-property 'layers-showing}
  old-below-current-layer:bool={!= 0 {bit-and layers-showing
                                              LAYERS-BELOW-CUR}}
  old-current-layer:bool={!= 0 {bit-and layers-showing LAYERS-CUR}}
  old-above-current-layer:bool={!= 0 {bit-and layers-showing
                                              LAYERS-ABOVE-CUR}}
```

```
current-depth:bool={self.get-application-property 'depth}

{cond {{and old-below-current-layer {not below-current-layer}}
       | need to hide all these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {< {x.get-object-property 'depth} current-depth}
                  {self.hide-canvas-object x}}
              }
            self.Objects}}
      {{and {not old-below-current-layer} below-current-layer}
       | need to show all of these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {< {x.get-object-property 'depth} current-depth}
                  {self.show-canvas-object x}}
              }
            self.Objects}
       }
      } | cond
{cond {{and old-current-layer {not current-layer}}
       | need to hide these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {= {x.get-object-property 'depth} current-depth}
                  {self.hide-canvas-object x}}
              }
            self.Objects}
       }
      {{and {not old-current-layer} current-layer}
       | need to show these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {= {x.get-object-property 'depth} current-depth}
                  {self.show-canvas-object x}}
              }
            self.Objects}
       }
      } | cond
{cond {{and old-above-current-layer {not above-current-layer}}
       | need to hide these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {> {x.get-object-property 'depth} current-depth}
                  {self.hide-canvas-object x}}
              }
            self.Objects}
       }
      {{and {not old-above-current-layer} above-current-layer}
       | need to show these items
       {map {lambda {x}  | where x is a NewCanvasObject
              {if {> {x.get-object-property 'depth} current-depth}
```

```
                {self.show-canvas-object x}}
              }
             self.Objects}
        }
       } | cond

  {set layers-showing 0}
  {if below-current-layer
      {set layers-showing {bit-or layers-showing LAYERS-BELOW-CUR}}}
  {if current-layer
      {set layers-showing {bit-or layers-showing LAYERS-CUR}}}
  {if above-current-layer
      {set layers-showing {bit-or layers-showing LAYERS-ABOVE-CUR}}}
  {self.change-application-property 'layers-showing layers-showing}
  } | method: hide-layers

{define {show-all-layers}:void
  {map {lambda {x}  | where x is a NewCanvasObject
         {self.show-canvas-object x}
         }
       self.hidden-objects}
  } | method: show-all-layers

{define {WindowEvent2snap e:WindowEvent}:Point
  snap-p:Point
  {self.get-application-property 'snap-on}
  {cond {{self.get-application-property 'snap-on}
         {let floatx:float floaty:float
              snap-size:Point={self.get-property 'snap-size}
           | need to do it manually to get rounding
           | (rather than truncating)
           {set floatx {/ {cast float e.x} snap-size.x}}
           {set floaty {/ {cast float e.y} snap-size.y}}
           {set snap-p {new Point {round floatx} {round floaty}}}
           } | let
         } | snap-on
        {else
         {set snap-p {new Point e.x e.y}}
         } | else
        } | cond
  {return snap-p}
  } | method: WindowEvent2snap


| ******* Methods pretty much passed to member variables *******

| ***** Methods for the CEH *****
```

```
{define {popup-MEH-dialog}:void
  {if {void? self.MEH}
      {self.error "No mode selected"}
      {self.MEH.popup-dialog-box}
      }
  } | method: popup-MEH-dialog


| ***** Methods for the properties *****

{define {add-canvas-property property:symbol value:any}:void
  {if {self.canvas-properties.has-property-local? property}
      {throw {new Exception
                  {format void "~p~p~s"
                          "CanvasSheet.set-canvas-property: "
                          "setting unknown property "
                          property}}}
      {self.canvas-properties.set-one-property property value}
      }
  } | method: add-canvas-property


{define {change-canvas-property property:symbol
                                 newvalue:any
                                 not-selected-objects:bool=false}:void
  temp-button:NewPropertiesToolbarButton

  {cond {{not {self.canvas-properties.has-property-local? property}}
         {throw {new Exception
                     {format void "~p~p~s"
                             "CanvasSheet.set-canvas-property: "
                             "setting unknown property "
                             property}}}
         }
        {{or {void? self.select-boxes} not-selected-objects}
         | Just change it in the canvas
         {self.canvas-properties.set-one-property property newvalue}
         {set temp-button
              {self.Container.Container.get-canvas-property-button
                 property}}
         {if {not {void? temp-button}}
             {temp-button.re-initialize}
             }
         }
        {else
         | Change the properties in each of the selected objects
```

```
        {map {lambda {x}    | where x is a NewCanvasObject
               {self.update-object-for-undo
                {typeof x}
                'change-object-property
                x
                commit=false
                property newvalue}
               }
             self.select-boxes}
        {self.commit-exchange}
        } | else
       } | cond
  } | method: change-canvas-property


{define {get-canvas-property property:symbol}:any
  {if {eq? property 'mode-centric} {breakpoint}}
  {if {not {self.canvas-properties.has-property-local? property}}
      {throw {new Exception
                  {format void "~p~p~s"
                          "CanvasSheet.get-canvas-property: "
                          "asking for unknown property "
                          property}}}
      {return {self.canvas-properties.get-property property}}
      }
  } | method: get-canvas-property


| ***** Methods that just pass stuff to the CanvasApplication *****
| This is so the CEH, etc, can get at these methods easily.  And,
| personally, I don't think that they should have to worry about
| being in a CanvasApplication.


{define {error txt:text}:void
  {self.Container.Container.set-error-msg t=txt}
  } | method: error


{define {message msg}:void
  {self.Container.Container.set-message t=msg}
  } | method: error


{define {set-event-descriptions pointer-press:text=""
                                pointer-dbl-clk:text=""
                                shiftify:text=""}:void
```

```
  {self.Container.Container.set-event-descriptions
   pointer-press=pointer-press
   shiftify=shiftify
   pointer-dbl-clk=pointer-dbl-clk
   }
  } | method: set-event-descriptions


{define {check-semaphore name:symbol}:bool
  {return {self.Container.Container.check-semaphore name}}
  } | method: check-semaphore
{define {lock-semaphore name:symbol}:bool
  {return {self.Container.Container.lock-semaphore name}}
  } | method: lock-semaphore
{define {release-semaphore name:symbol}:void
  {self.Container.Container.release-semaphore name}
  } | method: release-semaphore
{define {add-semaphore name:symbol}:void
  {self.Container.Container.add-semaphore name}
  } | method: add-semaphore

{define {add-application-property prop:symbol value:any}:void
  {self.Container.Container.add-application-property prop value}}
{define {get-application-property prop:symbol}:any
  {return {self.Container.Container.get-application-property prop}}}
{define {change-application-property prop:symbol newvalue:any}:void
  {self.Container.Container.change-application-property prop newvalue}}

| ******* Overwritten methods *******

| ***** Overwritten methods of EventHandler *****


{define public {key-press e:KeyEvent}:void
  {invoke-method EventHandler 'key-press self e}
  {if {void? self.MEH}
      {return}}
  {if {= e.key 27}   | esc
      {self.MEH.esc-key-press}
      {self.MEH.key-press e}
      }
  } | method: key-press


{define public {pointer-press e:PointerEvent}:void
  snap-p:Point
  gravity-proc:{proc {Point}:Point}
```

```
  temp-point:Point

  {invoke-method EventHandler 'pointer-press self e}
  | make sure that we're getting keyboard input
  {request-key-focus self}

  {if {void? self.MEH}
      {return}}
  {if {self.check-semaphore 'popup}
      {return}}
  {set snap-p {self.WindowEvent2snap e}}
  {set gravity-proc {self.get-application-property 'gravity-procedure}}
  {when {not {void? gravity-proc}}
    {set temp-point {new Point e.x e.y}}
    {set temp-point {gravity-proc temp-point}}
    {set e.x temp-point.x}
    {set e.y temp-point.y}
    }
  {self.MEH.pointer-press e snap-p}
  } | method: pointer-press


{define public {enter e:WindowEvent}:void
  snap-p:Point
  orig-p:Point

  {invoke-method EventHandler 'enter self e}
  {set snap-p {self.WindowEvent2snap e}}
  {set orig-p {new Point e.x e.y}}
  {self.Container.Hruler.update-enter orig-p snap-p}
  {self.Container.Vruler.update-enter orig-p snap-p}
  } | method: enter


{define public {leave e:WindowEvent}:void
  snap-p:Point
  orig-p:Point

  {invoke-method EventHandler 'leave self e}
  {set snap-p {self.WindowEvent2snap e}}
  {set orig-p {new Point e.x e.y}}
  {self.Container.Hruler.update-leave orig-p snap-p}
  {self.Container.Vruler.update-leave orig-p snap-p}
  } | method: leave


{define public {pointer-motion e:PointerEvent}:void
```

```
  snap-p:Point
  orig-p:Point

  {invoke-method EventHandler 'pointer-motion self e}
  {if {self.check-semaphore 'popup}
      {return}
      }
  {set snap-p {self.WindowEvent2snap e}}
  {set orig-p {new Point e.x e.y}}
  {if {not {void? self.MEH}}
      {self.MEH.pointer-motion e snap-p}}

  {self.Container.Hruler.update-value orig-p snap-p}
  {self.Container.Vruler.update-value orig-p snap-p}
  } | method: pointer-motion


{define public {pointer-release e:PointerEvent}:void
  snap-p:Point
  orig-p:Point

  {invoke-method EventHandler 'pointer-release self e}
  {set snap-p {self.WindowEvent2snap e}}
  {set orig-p {new Point e.x e.y}}

  {if {not {void? self.MEH}}
      {self.MEH.pointer-release e snap-p}}
  } | method: pointer-release


{define public {pointer-dbl-clk e:PointerEvent}:void
  snap-p:Point

  | don't call the super, it calls pointer-press
  | (and we don't need another...)
  {if {void? self.MEH}
      {return}}
  {set snap-p {self.WindowEvent2snap e}}
  {self.MEH.pointer-dbl-clk e snap-p}
  } | method: pointer-dbl-clk


| ***** Overwritten methods of Frame *****


{define public {invalidate}:void
  {invoke-method Frame 'invalidate self}
```

```
    {if {not {void? self.MEH}}
        {self.MEH.invalidate}
        }
    } | method: invalidate

  {define public {draw gc:GraphicContext}:void
    {invoke-method Frame 'draw self gc}
    {if {not {void? self.MEH}}
        {self.MEH.draw gc}
        }
    } | method: draw

  } | class: NewCanvasSheet


{define-class Grid {Frame}
  canvas:NewCanvasApplication
  width:int
  height:int
  showing:bool

  {define {init width:int height:int showing:bool=false}
    {super.init width=width height=height}
    {set self.width width}
    {set self.height height}
    {set self.showing showing}
    } | method: init

  {define {set-canvas canvas:NewCanvasApplication}:void
    {set self.canvas canvas}
    } | method: set-canvas

  {define {show}:void
    {set self.showing true}
    {self.invalidate}
    } | method: show

  {define {hide}:void
    {set self.showing false}
    {self.invalidate}
    } | method: hide

  {define {resize width:int=-1 height:int=-1}:void
    {when {!= width -1}
      {self.set-property width=width}
      {set self.width width}
      }
```

```
    {when {!= height -1}
      {self.set-property height=height}
      {set self.height height}
      }
    {self.invalidate}
    } | method: resize

  {define public {draw gc:GraphicContext}:void
    cur-x:int
    cur-y:int
    grid-size:Point

    {super.draw gc}
    {if {not self.showing}
        {return}
        }

    {set grid-size {self.canvas.get-application-property 'grid-size}}
    {set cur-y 0}
    {while {< cur-y self.height}
      {set cur-x 0}
      {while {< cur-x self.width}
        {gc.draw-line cur-x cur-y {+ cur-x 1} cur-y 'black 1}
        {set cur-x {+ cur-x grid-size.x}}
        } | inner-while
      {set cur-y {+ cur-y grid-size.y}}
      } | outer-while
    } | method: draw
  } | class: Grid

|#########################################################################
|#########################################################################
|#########################################################################
|#########################################################################

{define-class NewCanvasApplication {Frame}
  | the visual compenents
  modebar:DynamicToolbar
  propbar:DynamicToolbar
  menu:DynamicMenu
  error-msg-box:CanvasErrorBox
  event-desc-box:CanvasEventDescriptionBox
  canvas:CanvasAndRulers
  | non-visual
  canvas-sheets:DynamicRotateStructure
  num-scratch-sheets:int
  canvas-app-properties:CanvasPropertyList
```

```
semaphores:CanvasPropertyList | true means its locked

{define {init canvas:CanvasAndRulers width:int height:int}
  canvas-scrollbox:ScrollBox
  popup-dialog-button:Button
  toolbar-vbox:Vbox
  returned-graphic:Graphic
  modes-toolbar:NewModesToolbar
  prop-toolbar:NewPropertiesToolbar

  {super.init width=width height=height}

  {set self.modebar {new DynamicToolbar}}
  {set self.propbar {new DynamicToolbar}}
  {set self.error-msg-box {new CanvasErrorBox}}
  {set self.event-desc-box {new CanvasEventDescriptionBox}}
  {set self.canvas canvas}
  {set self.canvas-sheets {new DynamicRotateStructure}}
  {set self.num-scratch-sheets 1}
  {set self.canvas-app-properties {new CanvasPropertyList}}
  {set self.semaphores {new CanvasPropertyList}}
  {set self.menu {new DynamicMenu 'canvasapp}}
  {self.add-menu-heading void 'Window}

  {set modes-toolbar {new NewModesToolbar}}
  {modes-toolbar.set-canvas self}
  {self.modebar.add-toolbar 'draw modes-toolbar}

  {set prop-toolbar {new NewPropertiesToolbar}}
  {prop-toolbar.set-canvas self}
  {self.propbar.add-toolbar 'canvassheet prop-toolbar}

  {set popup-dialog-button
       {new Button label="Popup Dialog"
            action={action {self.canvas.Sheet.popup-MEH-dialog}}}}
  {{self.modebar.get-current-value}.set-canvas self}
  {{self.propbar.get-current-value}.set-canvas self}

  {set toolbar-vbox
       {spaced-vbox spacing={abs MODE-TOOLBAR-XPOS}
                    self.modebar
                    popup-dialog-button}}
  {self.append-object toolbar-vbox}
  {toolbar-vbox.set-position x=MODE-TOOLBAR-XPOS y=MODE-TOOLBAR-YPOS}

  {set returned-graphic {self.append-object self.menu}}
  {returned-graphic.set-position x=MENU-XPOS y=MENU-YPOS}
```

```
  {self.append-object self.error-msg-box}
  {self.error-msg-box.set-position x=ERROR-BOX-XPOS y=ERROR-BOX-YPOS}

  {self.append-object self.event-desc-box}
  {self.event-desc-box.set-position x=DESCRIPTION-BOX-XPOS
                                    y=DESCRIPTION-BOX-YPOS}

  {self.canvas.set-container self}
  {set canvas-scrollbox {new ScrollBox self.canvas}}
  {self.append-object canvas-scrollbox}
  {self.add-canvas-sheet self.canvas.Sheet 'scratch}
  {canvas-scrollbox.set-position x=CANVAS-SHEET-AND-RULERS-XPOS
                                 y=CANVAS-SHEET-AND-RULERS-YPOS}

  {self.append-object self.propbar}
  {self.propbar.set-position x=PROP-TOOLBAR-XPOS y=PROP-TOOLBAR-YPOS}
  } | method: init



{define {add-canvas-sheet sheet:NewCanvasSheet
                          name:symbol
                          add-on-top:bool=true}:bool
  {if {self.canvas-sheets.is-name? name}
      {return false}}
  {self.canvas-sheets.add-name name value=sheet}

  {self.add-menu-item
   {make-list 'Window}
   name
   action={canvas-action {self.set-current-sheet name}}
   }
  {if add-on-top
      {self.set-current-sheet name}}
  {return true}
  } | method: add-canvas-sheet


{define {set-current-sheet name:symbol}:void
  new-sheet:NewCanvasSheet
  old-sheet:NewCanvasSheet

  {set old-sheet self.canvas.Sheet}
  {self.canvas-sheets.set-current-name name}
  {set new-sheet {self.canvas-sheets.get-current-value}}
  {if {neq? old-sheet new-sheet}
```

```
    {self.canvas.change-sheet new-sheet}
    }
  } | method: set-current-sheet

{define {remove-canvas-sheet sheet:NewCanvasSheet=void
                             name:symbol=void}:void
  new-sheet-name:symbol

  {cond {{not {void? name}}
         {if {not {self.canvas-sheets.is-name? name}}
             {throw {new Exception
                         {format void "~p ~s"
                                 "Removing a CanvasSheet of unknown name"
                                 name}}}
             }
         {self.canvas-sheets.remove-name name}
         {self.remove-menu-item {make-list 'Window} name}
         } | cond-name
        {{not {void? sheet}}
         {set name {self.canvas-sheets.get-name sheet}}
         {if {void? name}
             {throw {new Exception
                         {format void "~p~p"
                                 "Removing a CanvasSheet not "
                                 "currently added"}}}
             }
         {self.canvas-sheets.remove-name name}
         {self.remove-menu-item {make-list 'Window} name}
         } | cond-sheet
        {else
         {return}
         }
        } | cond
  | now just change the current CanvasSheet.  Worse comes to
  | worse, we're setting it to the same thing.
  {set new-sheet-name {self.canvas-sheets.get-current-name}}
  {cond  {{void? sheet}
          | no more sheets left.  Need to make a new guy
          {self.make-new-sheet}
          } | cond void-sheet
         {else
          {self.set-current-sheet new-sheet-name}
          }} | else-cond
  } | method: remove-canvas-sheet

{define {make-new-sheet put-on-top:bool=true}:list
  width:int
```

```
  height:int
  sheet:NewCanvasSheet
  name:symbol

  {set width {self.get-application-property 'sheet-width}}
  {set height {self.get-application-property 'sheet-height}}
  {set sheet {new NewCanvasSheet width=width height=height}}
  {set self.num-scratch-sheets {+ self.num-scratch-sheets 1}}
  {set name {text->symbol
             {text-append "scratch"
                          {->text self.num-scratch-sheets}}}}
  {self.add-canvas-sheet sheet name add-on-top=put-on-top}
  {return {make-list sheet name}}
  } | method: make-new-sheet

| hack: can't have a symbol before the rest arg
{define {add-menu-heading path:list
                          heading-name:any
                          menu-name:symbol='canvasapp
                          void:int=3 ...}:void
  {generic-apply-method DynamicMenu 'add-sub-menu self.menu
                        {rest-as-list}
                        path heading-name menu-name=menu-name}
  } | method: add-menu-heading


| hack: can't have a symbol before the rest arg
{define {add-menu-item path:list
                       item-name:any
                       action:canvas-action-type=no-canvas-action
                       menu-name:symbol='canvasapp void:int=3 ...}:void
  passed-action:action-type

  {set passed-action
       {lambda {void}:void
         {cond {{or {self.check-semaphore 'mode}
                    {self.check-semaphore 'popup}}
                | can't do it
                {self.set-error-msg t="Menu currently disabled"}
                }
               {else
                {self.set-message t=""}
                {action self.canvas.Sheet}
                }} | else-cond
         } | lambda
       } | set
  {generic-apply-method DynamicMenu 'add-menu-action self.menu
```

```
                        {rest-as-list} path item-name
                        action=passed-action menu-name=menu-name}
  } | method: add-menu-item


{define {remove-menu-item path:list
                          item-name:any
                          menu-name:symbol='canvasapp}:void
  {self.menu.remove-menu-action path item-name menu-name=menu-name}
  } | method: remove-menu-item


{define {add-toolbar tbar-type:symbol
                     tbar-name:symbol
                     tbar:NewToolbar}:void
  {cond {{eq? tbar-name 'modes}
         {self.modebar.add-toolbar tbar-name tbar}
         }
        {{eq? tbar-type 'properties}
         {self.propbar.add-toolbar tbar-name tbar}
         }
        {else
         {error {format void "Unknown toolbar type ~s" tbar-name}}
         }
        } | cond
  } | method: add-toolbar


{define {change-toolbar tbar-type:symbol tbar-name:symbol}:void
  {cond {{eq? tbar-name 'modes}
         {self.modebar.change-dynamic-toolbar tbar-name}
         }
        {{eq? tbar-type 'properties}
         {self.propbar.change-dynamic-toolbar tbar-name}
         }
        {else
         {error {format void "Unknown toolbar type ~s" tbar-name}}
         }
        } | cond
  } | method: add-toolbar


{define {add-toolbar-button tbar-type:symbol
                            tbar-name:symbol
                            toolbar-button:NewToolbarButton
                            |ck: put next line in at some point
                            | button-name:symbol
```

```
                              row:int=-1 col:int=-1}:void
  {cond {{eq? tbar-type 'modes}
         {self.modebar.add-button tbar-name toolbar-button
                                  row=row col=col}
         }
        {{eq? tbar-type 'properties}
         {self.propbar.add-button tbar-name toolbar-button
                                  row=row col=col}
         }
        {else
         {error {format void "Unknown toolbar type ~s" tbar-name}}
         }
        } | cond
  } | method: add-toolbar-button


{define {set-error-msg t:Graphic=""}
  {self.error-msg-box.new-error error-msg=t}
  } | method: set-error-msg

{define {set-message t:Graphic=""}
  {self.error-msg-box.new-error error-msg=t}
  } | method: set-message

{define {set-event-descriptions pointer-press:text=""
                                pointer-dbl-clk:text=""
                                shiftify:text=""}:void
  {self.event-desc-box.new-descriptions
   pointer-press-desc=pointer-press
   shift-pointer-press-desc=shiftify
   pointer-dbl-clk-desc=pointer-dbl-clk
   }
  } | method: set-event-descriptions


{define {add-semaphore name:symbol}:void
  {if {self.semaphores.has-property-local? name}
      {throw {new Exception {format void "~p~s~p"
                                         "Adding semaphore "
                                         name
                                         ", which already exists."}}}
      }
  {self.semaphores.set-one-property name false}
  } | method: add-semaphore
{define {check-semaphore name:symbol}:bool
  {if {not {self.semaphores.has-property-local? name}}
      {throw {new Exception
```

```
                {format void "Checking unknown semaphore ~s" name}}}
      {return {self.semaphores.get-property name}}
      }
  } | method: check-semaphore
{define {lock-semaphore name:symbol}:bool
  {cond {{not {self.semaphores.has-property-local? name}}
         {throw {new Exception
                     {format void "Locking unknown semaphore ~s" name}}}
         }
        {{self.semaphores.get-property name}
         | already locked
         {return false}
         }
        {else
         | give it to 'em
         {self.semaphores.set-one-property name true}
         {return true}
         }
        } | cond
  } | method: lock-semaphore
{define {release-semaphore name:symbol}:void
  {if {not {self.semaphores.has-property-local? name}}
      {throw {new Exception
  {format void "Releasing unknown semaphore ~s" name}}}
      {self.semaphores.set-one-property name false}
      }
  } | method: release-semaphore


{define {set-canvas-property-button property:symbol
                                    button:NewPropertiesToolbarButton
    }:void
  new-property:symbol={text->symbol {text-append {->text property}
                                                 "-button"}}
  failed:bool=false
  {try
   {self.get-canvas-property property}
   {self.canvas-app-properties.set-one-property new-property button}
   {catch {e:Exception}
          {set failed true}}
   } | try
  {if failed
      {throw {new Exception {format void "~p~p~s"
                                         "set-canvas-property-button: "
                                         "unknown property "
                                         property}}}
      }
```

```
  } | set-canvas-property-button

{define {get-canvas-property-button
             property:symbol
             }:NewPropertiesToolbarButton
  new-property:symbol={text->symbol {text-append {->text property}
                                                 "-button"}}
  failed:bool=false
  ret-value:NewPropertiesToolbarButton
  {try
   {self.get-canvas-property property}
   {set ret-value {self.canvas-app-properties.get-property new-property}}
   {catch {e:Exception}
          {set failed true}}
   } | try
  {if failed
      {throw {new Exception {format void "~p~p~s"
                                         "get-canvas-property-button: "
                                         "unknown property "
                                         property}}}
      {return ret-value}
      }
  }

{define {add-canvas-property prop:symbol value:any}:void
  {self.canvas.Sheet.add-canvas-property prop value}
  } | method: add-canvas-property

{define {get-canvas-property prop:symbol}:any
  {return {self.canvas.Sheet.get-canvas-property prop}}
  } | method: get-canvas-property

{define {change-canvas-property prop:symbol
                                 newvalue:any
                                 not-selected-objects:bool=false}:void
  {self.canvas.Sheet.change-canvas-property
        prop newvalue not-selected-objects=not-selected-objects}
  } | method: change-canvas-property


{define {add-application-property prop:symbol value:any}:void
  {if {self.canvas-app-properties.has-property-local? prop}
      {throw {new Exception
                  {format void "~p~p~s"
                          "CanvasApp.add-application-property: "
                          "trying to re-add property "
```

```
                        prop}}}
      {self.canvas-app-properties.set-one-property prop value}
      }
  } | method: add-application-property

{define {get-application-property prop:symbol}:any
  {if {not {self.canvas-app-properties.has-property-local? prop}}
      {throw {new Exception
                  {format void "~p~p~s"
                          "CanvasApplication.get-application-property: "
                          "asking for unknown property "
                          prop}}}
      {begin
       {return {self.canvas-app-properties.get-property prop}}
       }
      }
  } | method: get-application-property

{define {change-application-property prop:symbol newvalue:any}:void
  {if {not {self.canvas-app-properties.has-property-local? prop}}
      {throw {new Exception
                  {format void "~p:~p~s"
                          "CanvasApplication.change-application-property"
                          "setting unknown property"
                          prop}}}
      {self.canvas-app-properties.set-one-property prop newvalue}
      }
  } | method: change-application-property

} | class: NewCanvasApplication
```

A.4.2 cruler.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| These are the rulers that go around a canvas.  In addition, the
| markers for the rulers are defined here.

{define-mixin-class NewRulerMarkerNonGraphic {}
  showing:bool
  graphic-object:Graphic   | The Graphic that contains the marker
  point-x:int              | x-Location the marker represents
  point-y:int              | y-Location the marker represents

  {define {init showing:bool=false}
    | graphic-object should contain the marker
```

```
    | showing should only be true if we've
    | already "entered" the canvas sheet.
    {set self.showing showing}
    }
  {define {set-container graphic-object:Graphic}:void
    {set self.graphic-object graphic-object}
    }
  {define public {set-marked-position p:Point}:void
    {error "set-marked-position not overwritten"}}
  {define public {show}:void
    {set self.showing true}
    } | method: show
  {define public {hide}:void
    {set self.showing false}
    } | method: hide
  {define public {invalidate}:void
    {return}}
  {define public {draw}:void
    {error "draw not overwritten"}}
  } | mixin-class: NewRulerMarkerNonGraphic



{define-class NewStraightRulerMarkerNonGraphic {NewRulerMarkerNonGraphic}
  height:int
  width:int
  horizontal:bool
  on-top-left:bool
  use-reg-coords:bool
  invalidated:bool

  | the points, if in their own Graphic
  private p1:Point
  private p2:Point
  private p3:Point

  {define {init horizontal:bool=true on-top-left:bool=true
                use-reg-coords:bool=true marker-offset:int=0}
    x1:int
    y1:int
    x2:int
    y2:int
    x3:int
    y3:int

    {super.init}
    {set self.height RULER-MARKER-HEIGHT}
```

```
{set self.width RULER-MARKER-WIDTH}
{set self.horizontal horizontal}
{set self.on-top-left on-top-left}
{if {not use-reg-coords}
    {if self.horizontal
        {set self.point-y {- marker-offset self.height}}
        {set self.point-x {- marker-offset self.height}}
        }
    }
{set self.invalidated false}

| some draw stuff, put here for speed
{cond {{and self.horizontal self.on-top-left}
       {set x1 0}
       {set y1 0}
       {set x2 {/ self.width 2}}
       {set y2 self.height}
       {set x3 self.width}
       {set y3 0}
       }
      {{and self.horizontal {not self.on-top-left}}
       {set x1 self.height}
       {set y1 0}
       {set x2 {/ self.width 2}}
       {set y2 0}
       {set x3 self.width}
       {set y3 self.height}
       }
      {{and {not self.horizontal} {not self.on-top-left}}
       {set x1 self.height}
       {set y1 0}
       {set x2 0}
       {set y2 {/ self.width 2}}
       {set x3 self.height}
       {set y3 self.width}
       }
      {else
       {set x1 0}
       {set y1 0}
       {set x2 self.height}
       {set y2 {/ self.width 2}}
       {set x3 0}
       {set y3 self.width}
       }
      } | cond
{set self.p1 {new Point x1 y1}}
{set self.p2 {new Point x2 y2}}
```

```
  {set self.p3 {new Point x3 y3}}
  } | method: init

{define public {set-marked-position p:Point}:void
  {if {and self.showing
           {not self.invalidated}}
      {self.draw}    | to erase old
      {set self.invalidated false}
      }

  {if self.horizontal
      {set self.point-x {- p.x {/ self.width 2}}}
      {set self.point-y {- p.y {/ self.width 2}}}
      }
  {if self.showing
      {self.draw}   | to draw new
      }
  } | method: set-marked-position

{define public {show}:void
  | don't need to draw, becacuse we'll get a mouse-motion
  {set self.showing true}
  } | method: show

{define public {hide}:void
  {self.draw}
  {set self.showing false}
  {set self.invalidated true}
  } | method: hide


{define public {invalidate}:void
  {set self.invalidated true}
  } | method: invalidate


{define public {draw}:void
  | Uses rubberband-line to draw a triangle that points to the (x,y)
  | coords.
  x1:int
  y1:int
  x2:int
  y2:int
  x3:int
  y3:int

  {if {not self.showing}
```

```
        {return}}
      {when self.invalidated
        {set self.invalidated false}
        {return}
        } | when

      | setup the triangle
      {set x1 {+ self.p1.x self.point-x}}
      {set x2 {+ self.p2.x self.point-x}}
      {set x3 {+ self.p3.x self.point-x}}
      {set y1 {+ self.p1.y self.point-y}}
      {set y2 {+ self.p2.y self.point-y}}
      {set y3 {+ self.p3.y self.point-y}}
      | and draw the triangle
      | bug: doesn't work right under linux
      {rubberband-line self.graphic-object x1 y1 x2 y2}
      {rubberband-line self.graphic-object x2 y2 x3 y3}
      {rubberband-line self.graphic-object x3 y3 x1 y1}
      } | method: draw

  } | class: NewStraightRulerMarkerNonGraphic


{define-mixin-class NewRuler {Frame}
  | Generic ruler for showing position.
  linecolor:any
  marker:NewRulerMarkerNonGraphic
  showing:bool

  | hack: can't have a symbol be the last thing before a ...
  {define {init position-marker:NewRulerMarkerNonGraphic
                showing:bool=false linecolor:any='black void:bool=true ...}
    {set self.marker position-marker}
    {set self.linecolor linecolor}
    {generic-apply-method Frame 'init self {rest-as-list} background='grey}
    {position-marker.set-container self}
    {set self.showing showing}
    } | method: init
  {define {update-value p:Point snap-p:Point}:void
    {error "update-value not overwritten"}}
  {define {update-enter p:Point snap-p:Point}:void
    {if self.showing
        {self.marker.show}}
    } | method: update-enter
  {define {update-leave p:Point snap-p:Point}:void
    {if self.showing
        {self.marker.hide}}
```

```
    } | method: update-leave

  {define public {draw gc:GraphicContext}:void
    {if self.showing
        {self.marker.invalidate}
        }
    {return}
    } | method: draw

  {define public {invalidate}:void
    {super.invalidate}
    {self.marker.invalidate}
    {return}
    } | method: invalidate

  {define public {change-marker
                  newmarker:NewRulerMarkerNonGraphic
                  }:NewRulerMarkerNonGraphic
    oldmarker:NewRulerMarkerNonGraphic
    {set oldmarker self.marker}
    {oldmarker.hide}
    {newmarker.hide}
    {set self.marker newmarker}
    {newmarker.show}
    {return oldmarker}
    } | method: change-marker
  } | mixin-class: NewRuler


{define-class NewStraightRuler {NewRuler}
  | A standard straight-edge ruler.

  | ck: All units are in pixels now.
  | Ask David if you can do inches easily.
  height:int
  width:int
  horizontal:bool
  on-top-left:bool
  foreground-color:any

  {define {init foreground-color:any='black
                height:int=RULER-HEIGHT width:int=RULER-WIDTH
                horizontal:bool=true on-top-left:bool=true
                border-style:symbol='black border-width:int=1}
    marker:NewStraightRulerMarkerNonGraphic

    {if {or {and horizontal on-top-left}
```

```
        {and {not horizontal} on-top-left}
        }
      {set marker {new NewStraightRulerMarkerNonGraphic
                       horizontal=horizontal on-top-left=on-top-left
                       use-reg-coords=false marker-offset=height}}
      {set marker {new NewStraightRulerMarkerNonGraphic
                       horizontal=horizontal on-top-left=on-top-left
                       use-reg-coords=true}}
      }
  {marker.set-container self}
  {if horizontal
      {super.init marker linecolor=foreground-color
                  height=height width=width
                  border-style=border-style border-width=border-width}
      {super.init marker linecolor=foreground-color
                  height=width width=height
                  border-style=border-style border-width=border-width}
      }
  {set self.height height}
  {set self.width width}
  {set self.horizontal horizontal}
  {set self.on-top-left on-top-left}
  {set self.foreground-color foreground-color}
  } | method: init

{define {update-value p:Point snap-p:Point}:void
  {self.marker.set-marked-position snap-p}
  } | method: update-value


{define public {draw gc:GraphicContext}:void
  offset:int
  long-position:int     | where the long lines start
  short-position:int    | where the short lines start
  number-of-large:int   | {/ width LARGE-SEP}
  number-of-small:int   | {/ width SMALL-SEP}, so it includes large

  tunit:text     | the text of the units for the ruler
  tunit-length:int  | number of characters
  tunit-position    | where the text starts
  tunit-offset      | where the text starts

  {gc.set-color self.foreground-color}

  | for efficiency
  {set long-position {/ self.height 2}}
  {cond {{or {and self.horizontal self.on-top-left}
```

```
          {and {not self.horizontal} self.on-top-left}}
       {set short-position {* 3 {/ self.height 4}}}
       {set tunit-position {/ self.height 2}}
       }
      {else
       {set short-position {/ self.height 4}}
       {set tunit-position {/ self.height 2}}
       }
      } | cond

| first, the large guys
{set offset 0}
{dotimes {i {/ self.width RULER-TICKS-LARGE-SEP}}
  {set offset {+ offset RULER-TICKS-LARGE-SEP}}
  {set tunit {->text i}}
  {set tunit-length {length tunit}}
  {if self.horizontal
      {set tunit-offset
           {- offset {/ {text-width tunit self} 2}}}
      {set tunit-offset
           {+ offset {/ {text-height self} 2}}}
      }

  {cond {{and self.horizontal self.on-top-left}
         | ruler above something
         {gc.draw-line offset long-position
                       offset self.height
                       self.linecolor 1}
         {gc.draw-text tunit-offset tunit-position
                       tunit 0 tunit-length}
         }
        {{and {not self.horizontal} {not self.on-top-left}}
         | ruler to the right of something
         {gc.draw-line long-position offset
                       0 offset
                       self.linecolor 1}
         {gc.draw-text tunit-position tunit-offset
                       tunit 0 tunit-length}
         }
        {{and {not self.horizontal} self.on-top-left}
         | ruler to the left of something
         {gc.draw-line long-position offset
                       self.height offset
                       self.linecolor 1}
         {gc.draw-text tunit-position tunit-offset
                       tunit 0 tunit-length}
         }
```

```
          {else
           | ruler below something
           {gc.draw-line offset long-position
                         offset 0
                         self.linecolor 1}
           {gc.draw-text tunit-offset tunit-position
                         tunit 0 tunit-length}
           }
          } | cond

    } | dotimes i

  | now the short guys.
  {set offset 0}
  {dotimes {j {/ self.width RULER-TICKS-SMALL-SEP}}
    {set offset {+ offset RULER-TICKS-SMALL-SEP}}
    {if {not {= {% offset RULER-TICKS-LARGE-SEP} 0}}
    {cond {{and self.horizontal self.on-top-left}
           {gc.draw-line offset short-position
                         offset self.height
                         self.linecolor 1}
           }
          {{and {not self.horizontal} {not self.on-top-left}}
           {gc.draw-line short-position offset
                         0 offset
                         self.linecolor 1}
           }
          {{and {not self.horizontal} self.on-top-left}
           {gc.draw-line short-position offset
                         self.height offset
                         self.linecolor 1}
           }
          {else
           {gc.draw-line offset short-position
                         offset 0
                         self.linecolor 1}
           }
          } | cond

        } | if not a large sep
    } | dotimes j

  | and finally, the super
  {super.draw gc}
  } | method: draw
} | class: NewStraightRuler
```

A.4.3 ctoolbar.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains the classes used to instantiate both Canvas
| toolbars and their buttons

{define-mixin-class NewCanvasToolbar {}
  canvas:NewCanvasApplication

  {define {init}
    {return}
    } | method: init
  {define {set-canvas canvas:NewCanvasApplication}:void
    {set self.canvas canvas}
    } | method: set-canvas
  {define {add-button b:NewCanvasToolbarButton row:int=-1 col:int=-1}
    {b.set-canvas self.canvas}
    } | method: add-button
  {define {get-canvas-sheet}:NewCanvasSheet
    {return self.canvas.canvas.Sheet}
    }
  } | mixin: NewCanvasToolbar


{define-mixin-class NewCanvasToolbarButton {}
  canvas:NewCanvasApplication

  {define {init}
    {return}
    } | method: init
  {define {set-canvas canvas:NewCanvasApplication}:void
    {set self.canvas canvas}
    } | method: set-canvas
  {define {get-canvas-sheet}:NewCanvasSheet
    {return self.canvas.canvas.Sheet}
    } | method: set-canvas
  } | mixin: NewCanvasToolbarButton


{define-class NewModesToolbar {NewRadioToolbar NewCanvasToolbar}
  edit-row:int
  draw-row:int
  action-row:int

  {define {init}
    {invoke-method NewRadioToolbar 'init self
```

```
                horizontal=false
                test={lambda {x:any}:bool
                       {return {isa NewModesToolbarButton x}}}
                spacing=0 cell-margin=0
                } | invoke-method
  {invoke-method NewCanvasToolbar 'init self}
  {set self.draw-row 1}
  {set self.edit-row 101}
  {set self.action-row 201}
  {invoke-method NewRadioToolbar 'add-object self
                {new Frame
                     {paragraph color='white DRAW}
                     width={* 2 MODE-BUTTON-WIDTH}
                     height=MODE-BUTTON-HEIGHT
                     border-width=1 border-style='black
                     background='black}
                0 0}
  {invoke-method NewRadioToolbar 'add-object self
                {new Frame
                     {paragraph color='white EDIT}
                     width={* 2 MODE-BUTTON-WIDTH}
                     height=MODE-BUTTON-HEIGHT
                     border-width=1 border-style='black
                     background='black}
                100 0}
  {invoke-method NewRadioToolbar 'add-object self
                {new Frame
                     {paragraph color='white ACTION}
                     width={* 2 MODE-BUTTON-WIDTH}
                     height=MODE-BUTTON-HEIGHT
                     border-width=1 border-style='black
                     background='black}
                200 0}
  } | method: init

{define {add-button button:NewToolbarButton
                    row:int=-1 col:int=-1}:void

  {if {= col -1}
      {set col 0}}
  {if {!= row -1}
      | handle this now, rather than 3 times below.  The else is below.
      {invoke-method NewRadioToolbar 'add-button self button
                     row=row col=col}}

  | keep going, for the update of self.(edit/draw/action)-row
  {cond {{isa NewEditModesToolbarButton button}
```

```
          {if {= row -1}
              {invoke-method NewRadioToolbar 'add-button self
                             button row=self.edit-row col=col}}
          {set self.edit-row {+ 1 self.edit-row}}
          }
         {{isa NewDrawModesToolbarButton button}
          {if {= row -1}
              {invoke-method NewRadioToolbar 'add-button self
                             button row=self.draw-row col=col}}
          {set self.draw-row {+ 1 self.draw-row}}
          }
         {{isa NewActionModesToolbarButton button}
          {if {= row -1}
              {invoke-method NewRadioToolbar 'add-button self
                             button row=self.action-row col=col}}
          {set self.action-row {+ 1 self.action-row}}
          }
         {else
          {throw {new Exception
                      {format void "Unknown button type ~p"
                              {typeof button}}}}
          } | else
         } | cond

    {invoke-method NewCanvasToolbar 'add-button self
      {cast NewCanvasToolbarButton button}}
    } | method add-button

  } | class NewModesToolbar


{define-mixin-class NewModesToolbarButton {NewCanvasToolbarButton}
  {define {init}
    {return}
    } | method: init
  } | mixin: NewModesToolbarButton


{define-class NewModesToolbarRadioButton {NewRadioToolbarButton
                                          NewModesToolbarButton}
  evhandler:NewCanvasEventHandler

  {define {init evhandler:NewCanvasEventHandler
                label:Graphic=void value:text=void}
    newlabel:Frame={new Frame
                        width=MODE-BUTTON-WIDTH
                        height=MODE-BUTTON-HEIGHT
```

```
                        label}
    {set self.evhandler evhandler}
    {invoke-method NewRadioToolbarButton 'init self
                   label=label
                   value=value
                   action={lambda {x:any}:void
                            {self.change-canvas-mode}
                            } | lambda
                   } | the call to super/NewRadioToolbarButton.init
    {invoke-method NewModesToolbarButton 'init self}
    } | method: init

  {define {set-canvas canvas:NewCanvasApplication}:void
    {invoke-method NewModesToolbarButton 'set-canvas self canvas}
    {self.evhandler.set-canvas canvas}
    } | set-canvas

  {define {change-canvas-mode}:void
    {{self.get-canvas-sheet}.change-MEH self.evhandler}
    } | method change-canvas-mode

  {define public {pointer-press e:PointerEvent}:void
    canvas:NewCanvasSheet={self.get-canvas-sheet}
    {if {or {canvas.check-semaphore 'mode} {canvas.check-semaphore 'popup}}
        | can't do it
        {{self.get-canvas-sheet}.message "Button disabled"}
         {invoke-method NewRadioToolbarButton 'pointer-press self e}
         }
    } | method: pointer-press

  } | class: NewModesToolbarRadioButton


{define-class NewModesToolbarNonRadioButton {NewToolbarButton
                                             NewModesToolbarButton}
  evhandler:NewCanvasEventHandler

  {define {init evhandler:NewCanvasEventHandler
                label:Graphic=void value:text=void
                action:action-type=no-action}
    newlabel:Frame={new Frame
                        width=MODE-BUTTON-WIDTH
                        height=MODE-BUTTON-HEIGHT
                        label}
    {set self.evhandler evhandler}
    {invoke-method NewToolbarButton 'init self
      label=label
```

```
      value=value
      action=action
      } | the call to super/NewRadioToolbarButton.init
    {invoke-method NewModesToolbarButton 'init self}
    } | method: init

  {define public {pointer-press e:PointerEvent}:void
    canvas:NewCanvasSheet={self.get-canvas-sheet}
    {if {or {canvas.check-semaphore 'mode} {canvas.check-semaphore 'popup}}
        | can't do it
        {return}
        {invoke-method NewRadioToolbarButton 'pointer-press self e}
        }
    } | method: pointer-press

  } | class: NewModesToolbarNonRadioButton


{define-class NewDrawModesToolbarButton {NewModesToolbarRadioButton}
  {define {init ...}
    {myapply-method NewModesToolbarRadioButton 'init self {rest-as-list}}
    } | method: init
  } | class: NewDrawModesToolbarButton

{define-class NewEditModesToolbarButton {NewModesToolbarRadioButton}
  {define {init ...}
    {myapply-method NewModesToolbarRadioButton 'init self {rest-as-list}}
    } | method: init
  } | class: NewEditModesToolbarButton

{define-class NewActionModesToolbarButton {NewModesToolbarNonRadioButton}
  {define {init ...}
    {myapply-method NewModesToolbarNonRadioButton
                    'init self {rest-as-list}}
    } | method: init
  } | class: NewActionModesToolbarButton

|#########################################################################
|#########################################################################
|#########################################################################
|#########################################################################

{define-class NewPropertiesToolbar {NewToolbar NewCanvasToolbar}
  {define {init}
    {invoke-method NewToolbar 'init self
        test={lambda {x:any}:bool
               {return {isa NewPropertiesToolbarButton x}}}}
```

```
    {invoke-method NewCanvasToolbar 'init self}
    } | method: init

  {define {add-button b:NewToolbarButton
                      row:int=-1 col:int=-1}:void
    {invoke-method NewToolbar 'add-button self
                   b row=row col=col}
    {invoke-method NewCanvasToolbar 'add-button self
      {cast NewCanvasToolbarButton b}}
    } | method: add-button

  } | class: NewPropertiesToolbar


{define-class NewPropertiesToolbarButton {NewToolbarButton
                                           NewCanvasToolbarButton}
  dialog-proc:action-type

  {define {init label:PropertyButtonGraphic=void tooltip:Graphic=void
               dialog-proc:action-type=no-action}
    {invoke-method NewToolbarButton 'init self
                   label={cast Graphic label} value=tooltip
                   action={action
                           {if {self.can-popup-dialog?}
                               {self.popup-dialog}
                               }}
                   }
    {set self.dialog-proc dialog-proc}
    {invoke-method NewCanvasToolbarButton 'init self}
    {label.set-container self}
    } | method: init


  {define public {draw gc:GraphicContext}:void
    {super.draw gc}
    } | method: draw

  {define {can-popup-dialog?}:bool
    {return {not {{self.get-canvas-sheet}.check-semaphore 'popup}}}
    } | method: can-popup-dialog?

  {define {popup-dialog}:void
    canvas:NewCanvasSheet={self.get-canvas-sheet}
    {when {not {canvas.lock-semaphore 'popup}}
        | FAILED
        {error {format void "~p~p~p"
                       "Tried to call "
```

```
                    "NewPropertiesToolbarButton.popup-dialog "
                    "when not allowed."}}
      {return}
      }
    {self.dialog-proc self}
    {canvas.release-semaphore 'popup}
    } | method: popup-dialog

  {define {re-initialize}:void
    {self.invalidate}
    } | re-initialize
  } | class: NewPropertiesToolbarButton
```

A.4.4 dialogs.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| These are all of the procedures that pull up dialog-boxes for
| user-input.

{define {rotation-degrees-property-dialog canvas:NewCanvasSheet}:void
  input-text:text
  input-int:int
  temp-list:list
  initial-value:int
  num-chars-parsed:int

  {set initial-value {canvas.Container.Container.get-application-property
                      'rotation-degrees}}
  {set input-text {->text initial-value}}
  {set input-text
       {popup-text-query
        size=10 prefix=input-text
        {new Verbatim valign='bottom
             {format void
                     "~a~%~a~%~a"
                     "Please specify an angle in degrees, intergers only."
                     "Positive angle is clock-wise"
                     "0 (the number zero) represents free rotation"}}
        }}
  {if {void? input-text}
      {return}}
  {set temp-list
       {text->int input-text return-length=true return-void=true}}
  {if {void? temp-list}
      {error "Invalid entry.  Ignoring"}}
```

```
  {set input-int {car temp-list}}
  {set num-chars-parsed {cadr temp-list}}
  {if {= {length input-text} num-chars-parsed}
      | It was a proper entry
      {canvas.Container.Container.change-application-property
         'rotation-degrees input-int}
      {error "Invalid entry.  Ignoring"}
      }
  } | procedure: rotation-degrees-property-dialog


{define {mode-centric-property-dialog canvas:NewCanvasSheet}:void
  first-button:RadioButton
  second-button:RadioButton
  r-buttons:RadioButtons

  mode-centric:bool={canvas.Container.Container.get-application-property
                       'mode-centric}
  result:symbol

  {cond {mode-centric
         | This is like xfig
         {set first-button
              {new RadioButton label="Tool remains selected"
                   pressed?=true}}
         {set second-button
              {new RadioButton label="Select-mode becomes selected"}}
         }
        {else
         {set first-button
              {new RadioButton label="Tool remains selected"}}
         {set second-button
              {new RadioButton label="Select-mode becomes selected"
                   pressed?=true}}
         } | else
        } | cond

  {set r-buttons
       {new RadioButtons action={lambda {button:any}:void
                                   {if {eq? button first-button}
                                       {set mode-centric true}
                                       {set mode-centric false}
                                       }}
            first-button second-button}}

  {set result {popup-dialogue r-buttons}}
  {if {eq? result 'ok}
```

```
      {canvas.Container.Container.change-application-property
         'mode-centric mode-centric}
      }
  } | procedure: mode-centric-property-dialog


{define {line-mode-dialog meh:LineModeEventHandler}:void
  {error "Dialog not ready for use."}
  } | procedure: line-mode-dialog


{define {line-color-dialog but:any}:void
  input-text:text
  input-int:int
  new-sym:symbol
  initial-value:any
  num-chars-parsed:int

  {set initial-value {but.canvas.get-canvas-property 'line-color}}
  {set input-text {color->text initial-value quote-sym=false}}
  {set input-text
       {popup-text-query
        size=10 prefix=input-text
        {new Verbatim valign='bottom
             "Please specify a color symbol."}}}
  {if {void? input-text}
      {return}}
  {set new-sym {text->symbol input-text}}
  {if {void? new-sym}
      {error "Invalid entry.  Ignoring"}}
  | check if that's a color
  {any-to-color new-sym}
  | It was a proper entry
  {but.canvas.change-canvas-property 'line-color new-sym}
  } | line-color-dialog


{define {depth-dialog canvas:NewCanvasSheet}:void
  input-text:text
  input-int:int
  chars-read:int
  initial-value:int
  temp-list:list

  {set initial-value {canvas.get-application-property 'depth}}
  {set input-text
       {popup-text-query
```

```
        size=10 prefix={->text initial-value}
        {new Verbatim valign='bottom
             {format void "~p~p"
                     "Please specify a positive, integer, depth.  "
                     "Large numbers are deeper."}}}}
  {if {void? input-text}
      {return}}
  {set temp-list {text->int input-text return-length=true}}
  {set input-int {car temp-list}}
  {set chars-read {cadr temp-list}}
  {if {!= {length input-text} chars-read}
      {error "Invalid entry.  Ignoring"}
      }
  {canvas.change-application-property 'depth input-int}
  } | depth-dialog


{define {line-width-dialog but:any}:void
  input-text:text
  input-int:int
  chars-read:int
  initial-value:int
  temp-list:list

  {set initial-value {but.canvas.get-canvas-property 'line-width}}
  {set input-text
       {popup-text-query
        size=10 prefix={->text initial-value}
        {new Verbatim valign='bottom
             "Please specify non-negative, integer, line-width."}}}
  {if {void? input-text}
      {return}}
  {set temp-list {text->int input-text return-length=true}}
  {set input-int {car temp-list}}
  {set chars-read {cadr temp-list}}
  {if {or {!= {length input-text} chars-read} {> 0 input-int}}
      {error "Invalid entry.  Ignoring"}
      }
  {but.canvas.change-canvas-property 'line-width input-int}
  } | depth-dialog


{define {fill-color-dialog but:any}:void
  input-text:text
  input-int:int
  new-sym:symbol
  initial-value:any
```

```
  num-chars-parsed:int

  {set initial-value {but.canvas.get-canvas-property 'fill-color}}
  {set input-text {color->text initial-value quote-sym=false}}
  {set input-text
       {popup-text-query
        size=10 prefix=input-text
        {new Verbatim valign='bottom
             "Please specify a color symbol.  Use \"none\" for no fill."}}}
  {if {void? input-text}
      {return}}
  {set new-sym {text->symbol input-text}}
  {if {void? new-sym}
      {error "Invalid entry.  Ignoring"}}
  | check if that's a color
  {if {or {eq? new-sym 'none} {eq? new-sym 'void}}
      {set new-sym void}
      {any-to-color new-sym}}
  | It was a proper entry
  {but.canvas.change-canvas-property 'fill-color new-sym}
  } | fill-color-dialog
```

A.4.5 message.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| Defines windows for displaying stuff.  They are specific to Canvas.

{define-class CanvasErrorBox {Frame}
  error-msg:Dynamic

  {define {init error-msg:Graphic=""
                width:int=ERROR-BOX-WIDTH
                height:int=ERROR-BOX-HEIGHT}
    tempGraphic:Graphic
    {super.init width=width height=height
                border-style='black border-width=1}
    {set self.error-msg {new Dynamic error-msg}}
    {set tempGraphic {self.append-object self.error-msg}}
    {tempGraphic.set-position x=5 y=5}
    } | method: init

  {define {new-error error-msg=""}
    {self.error-msg.set-value error-msg}
    } | method: new-error
  } | class: CanvasErrorBox
```

```
{define-class CanvasEventDescriptionBox {Frame}
  pointer-press-desc:Dynamic
  shift-pointer-press-desc:Dynamic
  pointer-dbl-clk-desc:Dynamic

  {define {init   pointer-press-desc:text=""
                  shift-pointer-press-desc:text=""
                  pointer-dbl-clk-desc:text=""
                  width:int=DESCRIPTION-BOX-WIDTH
                  height:int=DESCRIPTION-BOX-HEIGHT}
    temptext:text
    returned-graphic:Graphic

    {super.init width=width height=height
                border-style='black border-width=1 font-size=12}
    {set temptext "Pointer Press:"}
    {set returned-graphic {self.append-object temptext}}
    {returned-graphic.set-position x=5 y=5}
    {set temptext "With Shift:"}
    {set returned-graphic {self.append-object temptext}}
    {returned-graphic.set-position x=20 y=25}
    {set temptext "Double Click:"}
    {set returned-graphic {self.append-object temptext}}
    {returned-graphic.set-position x=5 y=45}

    {set self.pointer-press-desc {new Dynamic pointer-press-desc}}
    {set self.shift-pointer-press-desc
         {new Dynamic shift-pointer-press-desc}}
    {set self.pointer-dbl-clk-desc {new Dynamic pointer-dbl-clk-desc}}

    {set returned-graphic {self.append-object self.pointer-press-desc}}
    {returned-graphic.set-position x=85 y=5}
    {set returned-graphic
         {self.append-object self.shift-pointer-press-desc}}
    {returned-graphic.set-position x=85 y=25}
    {set returned-graphic {self.append-object self.pointer-dbl-clk-desc}}
    {returned-graphic.set-position x=85 y=45}
    } | method:init

  {define {new-descriptions pointer-press-desc:text=""
                            shift-pointer-press-desc:text=""
                            pointer-dbl-clk-desc:text=""}
    {self.pointer-press-desc.set-value pointer-press-desc}
    {self.shift-pointer-press-desc.set-value shift-pointer-press-desc}
    {self.pointer-dbl-clk-desc.set-value pointer-dbl-clk-desc}
```

```
    } | method: new-descriptions

  } | class: CanvasEventDescriptionBox
```

A.4.6 widget.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file contains the little graphics used in the canvas.  These
| include toolbar graphics and dialog graphics.

{define-class PropertyButtonGraphic {SimpleGraphic}
  button:NewPropertiesToolbarButton

  {define {init}
    {super.init}}
  {define public {layout}:{return int int}
    {return PROP-BUTTON-WIDTH PROP-BUTTON-HEIGHT}}
  {define {set-container b:NewPropertiesToolbarButton}:void
    {set self.button b}
    }
  {define {get-canvas-sheet}:NewCanvasSheet
    {return {self.button.get-canvas-sheet}}
    }
  }


{define-class ModeButtonGraphic {SimpleGraphic}
  button:NewPropertiesToolbarButton

  {define {init}
    {super.init}}

  {define public {layout}:{return int int}
    {return MODE-BUTTON-WIDTH MODE-BUTTON-HEIGHT}}
  }


{define-class LineColorButtonGraphic {PropertyButtonGraphic}
  {define {init}
    {super.init}
    }

  {define public {draw gc:GraphicContext}:void
    line-color:any
```

```
    {super.draw gc}
    {set line-color
         {{self.get-canvas-sheet}.get-canvas-property 'line-color}}
    {gc.draw-line 0 0 {self.graphic-get-width} {self.graphic-get-height}
                  line-color 1}
    }
  }

{define-class FillColorButtonGraphic {PropertyButtonGraphic}
  {define {init}
    {super.init}
    }

  {define public {draw gc:GraphicContext}:void
    fill-color:any
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}

    {super.draw gc}
    {set fill-color
         {{self.get-canvas-sheet}.get-canvas-property 'fill-color}}
    {if {not {void? fill-color}}
        {gc.fill-rect {/ width 4} {/ height 4}
                      {* 3 {/ width 4}} {* 3 {/ height 4}}
                      fill-color}
        {gc.draw-rect {/ width 4} {/ height 4}
                      {* 3 {/ width 4}} {* 3 {/ height 4}}
                      {self.get-property 'background} 'black 1}
        }
    }
  }

{define-class LineWidthButtonGraphic {PropertyButtonGraphic}
  {define {init}
    {super.init}
    }

  {define public {draw gc:GraphicContext}:void
    line-width:int={{self.get-canvas-sheet}.get-canvas-property
                        'line-width}
    line-width-text:text={->text line-width}
    center-x:int={/ {self.graphic-get-width} 2}
    center-y:int={/ {self.graphic-get-height} 2}
    offset-x:int={centering-offset line-width-text container=self}
    offset-y:int={- {centering-offset line-width-text
                                      container=self horizontal=false}
                    2}
```

```
    {super.draw gc}
    {gc.draw-line 0 center-y {- center-x offset-x} center-y
                  'black line-width}
    {gc.draw-line {+ center-x offset-x} center-y
                  {self.graphic-get-width} center-y
                  'black line-width}
    {gc.draw-text {- center-x offset-x} {+ center-y offset-y}
                  line-width-text 0 {length line-width-text}}
    } | method: draw
  } | class: LineWidthButton


{define-class PolyLineModeButtonGraphic {ModeButtonGraphic}
  {define {init}
    {super.init}}

  {define public {draw gc:GraphicContext}:void
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}
    one-tenth-width:int={/ width 10}
    one-tenth-height:int={/ height 10}
    p1:Point
    p2:Point
    p3:Point
    p4:Point
    p5:Point
    p6:Point

    {set p1 {new Point {* 4 one-tenth-width} {* 9 one-tenth-height}}}
    {set p2 {new Point {* 1 one-tenth-width} {* 7 one-tenth-height}}}
    {set p3 {new Point {* 1 one-tenth-width} {* 1 one-tenth-height}}}
    {set p4 {new Point {* 4 one-tenth-width} {* 4 one-tenth-height}}}
    {set p5 {new Point {* 9 one-tenth-width} {* 1 one-tenth-height}}}
    {set p6 {new Point {* 7 one-tenth-width} {* 6 one-tenth-height}}}
    {gc.draw-line p1.x p1.y p2.x p2.y 'black 1}
    {gc.draw-line p2.x p2.y p3.x p3.y 'black 1}
    {gc.draw-line p3.x p3.y p4.x p4.y 'black 1}
    {gc.draw-line p4.x p4.y p5.x p5.y 'black 1}
    {gc.draw-line p5.x p5.y p6.x p6.y 'black 1}
    }
  }

{define-class RectangleModeButtonGraphic {ModeButtonGraphic}
  {define {init}
    {super.init}}
```

```
  {define public {draw gc:GraphicContext}:void
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}
    one-tenth-width:int={/ width 10}
    one-tenth-height:int={/ height 10}

    {super.draw gc}
    {gc.draw-rect one-tenth-width one-tenth-height
                  {- width one-tenth-width} {- height one-tenth-height}
                  {self.get-property 'background} 'black 1}

    }
  }

{define-class ResizeModeButtonGraphic {ModeButtonGraphic}
  {define {init}
    {super.init}}

  {define public {draw gc:GraphicContext}:void
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}
    one-tenth-width:int={/ width 10}
    one-tenth-height:int={/ height 10}
    p1:Point p2:Point p3:Point
    p4:Point p5:Point p6:Point
    p7:Point p8:Point p9:Point
    p10:Point p11:Point p12:Point

    {set p1 {new Point {* 1 one-tenth-width} {* 3 one-tenth-height}}}
    {set p2 {new Point {* 0 one-tenth-width} {* 7 one-tenth-height}}}
    {set p3 {new Point {* 2 one-tenth-width} {* 7 one-tenth-height}}}

    {set p4 {new Point {* 2.5 one-tenth-width} {* 5 one-tenth-height}}}
    {set p5 {new Point {* 3.2 one-tenth-width} {* 3 one-tenth-height}}}
    {set p6 {new Point {* 3.2 one-tenth-width} {* 7 one-tenth-height}}}

    {set p7 {new Point {* 4.5 one-tenth-width} {* 5 one-tenth-height}}}
    {set p8 {new Point {* 3.8 one-tenth-width} {* 3 one-tenth-height}}}
    {set p9 {new Point {* 3.8 one-tenth-width} {* 7 one-tenth-height}}}

    {set p10 {new Point {* 7 one-tenth-width} {* 0 one-tenth-height}}}
    {set p11 {new Point {* 4.5 one-tenth-width} {* 9 one-tenth-height}}}
    {set p12 {new Point {* 9.5 one-tenth-width} {* 9 one-tenth-height}}}

    {gc.draw-line p1.x p1.y p2.x p2.y 'black 1}
    {gc.draw-line p2.x p2.y p3.x p3.y 'black 1}
    {gc.draw-line p3.x p3.y p1.x p1.y 'black 1}
```

```
    {gc.draw-line p4.x p4.y p5.x p5.y 'black 1}
    {gc.draw-line p4.x p4.y p6.x p6.y 'black 1}
    {gc.draw-line p4.x p4.y p7.x p7.y 'black 1}
    {gc.draw-line p7.x p7.y p8.x p8.y 'black 1}
    {gc.draw-line p7.x p7.y p9.x p9.y 'black 1}

    {gc.draw-line p10.x p10.y p11.x p11.y 'black 1}
    {gc.draw-line p11.x p11.y p12.x p12.y 'black 1}
    {gc.draw-line p12.x p12.y p10.x p10.y 'black 1}
    } | draw
  }


{define-class MoveModeButtonGraphic {ModeButtonGraphic}
  {define {init}
    {super.init}}

  {define public {draw gc:GraphicContext}:void
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}
    one-tenth-width:int={/ width 10}
    one-tenth-height:int={/ height 10}
    p1:Point p2:Point
    p3:Point p4:Point p5:Point
    p6:Point p7:Point p8:Point
    p9:Point p10:Point

    {set p1 {new Point {* 0 one-tenth-width} {* 2 one-tenth-height}}}
    {set p2 {new Point {* 3 one-tenth-width} {* 9 one-tenth-height}}}

    {set p3 {new Point {* 3.3 one-tenth-width} {* 5 one-tenth-height}}}
    {set p4 {new Point {* 4.5 one-tenth-width} {* 3 one-tenth-height}}}
    {set p5 {new Point {* 4.5 one-tenth-width} {* 7 one-tenth-height}}}

    {set p6 {new Point {* 6.2 one-tenth-width} {* 5 one-tenth-height}}}
    {set p7 {new Point {* 5 one-tenth-width} {* 3 one-tenth-height}}}
    {set p8 {new Point {* 5 one-tenth-width} {* 7 one-tenth-height}}}

    {set p9 {new Point {* 7 one-tenth-width} {* 2 one-tenth-height}}}
    {set p10 {new Point {* 10 one-tenth-width} {* 9 one-tenth-height}}}

    {gc.draw-rect p1.x p1.y p2.x p2.y
                  {self.get-property 'background} 'black 1}
    {gc.draw-line p3.x p3.y p4.x p4.y 'black 1}
    {gc.draw-line p3.x p3.y p5.x p5.y 'black 1}
    {gc.draw-line p3.x p3.y p6.x p6.y 'black 1}
```

```
    {gc.draw-line p6.x p6.y p7.x p7.y 'black 1}
    {gc.draw-line p6.x p6.y p8.x p8.y 'black 1}
    {gc.draw-rect p9.x p9.y p10.x p10.y
                  {self.get-property 'background} 'black 1}
    }
  }

{define-class RotateModeButtonGraphic {ModeButtonGraphic}
  {define {init}
    {super.init}}

  {define public {draw gc:GraphicContext}:void
    width:int={self.graphic-get-width}
    height:int={self.graphic-get-height}
    one-tenth-width:int={/ width 10}
    one-tenth-height:int={/ height 10}
    p1:Point p2:Point
    p3:Point p4:Point   p5:Point  p6:Point
    p7:Point  p8:Point  p9:Point  p10:Point
    p11:Point p12:Point p13:Point p14:Point
    p15:Point p16:Point p17:Point p18:Point

    {set p1 {new Point {* 2 one-tenth-width} {* 2 one-tenth-height}}}
    {set p2 {new Point {* 8 one-tenth-width} {* 8 one-tenth-height}}}

    {set p3 {new Point {* 4 one-tenth-width} {* 0 one-tenth-height}}}
    {set p4 {new Point {* 1 one-tenth-width} {* 2 one-tenth-height}}}
    {set p5 {new Point {* 1 one-tenth-width} {* 0 one-tenth-height}}}
    {set p6 {new Point {* 3 one-tenth-width} {* 2 one-tenth-height}}}

    {set p7 {new Point {* 1 one-tenth-width} {* 7 one-tenth-height}}}
    {set p8 {new Point {* 4 one-tenth-width} {* 9 one-tenth-height}}}
    {set p9 {new Point  {* 4 one-tenth-width}  {* 7 one-tenth-height}}}
    {set p10 {new Point {* 2 one-tenth-width}  {* 9 one-tenth-height}}}

    {set p11 {new Point {* 6 one-tenth-width} {* 9 one-tenth-height}}}
    {set p12 {new Point {* 9 one-tenth-width} {* 7 one-tenth-height}}}
    {set p13 {new Point {* 9 one-tenth-width} {* 9 one-tenth-height}}}
    {set p14 {new Point {* 7 one-tenth-width} {* 7 one-tenth-height}}}

    {set p15 {new Point {* 9 one-tenth-width} {* 2 one-tenth-height}}}
    {set p16 {new Point {* 6 one-tenth-width} {* 0 one-tenth-height}}}
    {set p17 {new Point {* 6 one-tenth-width} {* 2 one-tenth-height}}}
    {set p18 {new Point {* 8 one-tenth-width} {* 0 one-tenth-height}}}

    {gc.draw-ellipse p1.x p1.y p2.x p2.y
                     {self.get-property 'background} 'black 2}
```

```
    {gc.draw-line p3.x p3.y p4.x p4.y 'black 1}
    {gc.draw-line p4.x p4.y p5.x p5.y 'black 1}
    {gc.draw-line p4.x p4.y p6.x p6.y 'black 1}

    {gc.draw-line p7.x p7.y p8.x p8.y 'black 1}
    {gc.draw-line p8.x p8.y p9.x p9.y 'black 1}
    {gc.draw-line p8.x p8.y p10.x p10.y 'black 1}

    {gc.draw-line p11.x p11.y p12.x p12.y 'black 1}
    {gc.draw-line p12.x p12.y p13.x p13.y 'black 1}
    {gc.draw-line p12.x p12.y p14.x p14.y 'black 1}

    {gc.draw-line p15.x p15.y p16.x p16.y 'black 1}
    {gc.draw-line p16.x p16.y p17.x p17.y 'black 1}
    {gc.draw-line p16.x p16.y p18.x p18.y 'black 1}
    }
  }
```

A.5 The Canvas Application

A.5.1 all.curl

```
| File for CanvasApplication thesis, by Arthur Housinger

{include "v1.curl"}

{let app={new CanvasApp-v1-0}
     {value app}
  }
```

A.5.2 v1.curl

```
| File for CanvasApplication thesis, by Arthur Housinger
| This file makes sure everything is defined.  It then pulls up a
| sample CanvasApplication.

| import would be the proper thing, but it don't work.  (And I also
| don't feel like putting in all of the "public" stuff...

{include "type.curl"}
{include "common.curl"}
{include "generic.curl"}
```

```
{include "toolbar.curl"}

{include "widget.curl"}
{include "constants.curl"}
{include "geom.curl"}

{include "dynamic.curl"}
{include "menu.curl"}
{include "message.curl"}

{include "ctoolbar.curl"}

{include "cruler.curl"}
{include "cobjects.curl"}
{include "cevhand.curl"}

{include "markers.curl"}
{include "cbox.curl"}

{include "dialogs.curl"}
{include "cmenu.curl"}

{include "canvas.curl"}

{include "edit.curl"}
{include "line.curl"}
{include "rect.curl"}

{include "visible.curl"}

{define-class CanvasApp-v1-0 {NewCanvasApplication}
  {define {init
           | can place "flags" here
           Application-Width:int=CANVAS-APP-WIDTH
           Application-Height:int=CANVAS-APP-HEIGHT
           with-rulers:bool=true
           vertical-ruler-on-right:bool=true
           Sheet-Width:int=CANVAS-SHEET-WIDTH
           Sheet-Height:int=CANVAS-SHEET-HEIGHT
           Ruler-Height=RULER-HEIGHT
           }

    canvas-plus:2DCanvasAndRulers

    tempheight:int
    tempwidth:int
```

```
| setup the canvas-and-rulers
{set canvas-plus {new 2DCanvasAndRulers
                      rulers?=with-rulers
                      width={+ Sheet-Width Ruler-Height}
                      height={+ Sheet-Height Ruler-Height}
                      vert-ruler-on-right=vertical-ruler-on-right
                      ruler-height=Ruler-Height}}

| set up the canvas-application
{super.init canvas-plus Application-Width Application-Height}

| add application properties
{self.add-application-property 'depth 1}
{self.add-application-property 'degree-rotate 0}
{self.add-application-property 'ngon-sides 6}
{self.add-application-property 'alignment 0}
{self.add-application-property 'distribution 0}
{self.add-application-property 'sheet-width Sheet-Width}
{self.add-application-property 'sheet-height Sheet-Height}
{self.add-application-property 'mode-centric true}
{self.add-application-property 'show-rulers with-rulers}
{self.add-application-property 'draw-from-corner true}
{self.add-application-property 'snap-on false}
{self.add-application-property 'grid-on false}
{self.add-application-property 'snap-size {new Point 1 1}}
{self.add-application-property 'grid-size {new Point 5 5}}
{self.add-application-property 'gravity-procedure void}
{self.add-application-property 'copy-fresh true}
{self.add-application-property 'shift-angle 45}
{self.add-application-property 'nudge-size 5}
{self.add-application-property 'unit-type 'pixels}
{self.add-application-property 'large-ruler-tick-sep 100}
{self.add-application-property 'small-ruler-tick-amt 10}
{self.add-application-property 'arrow-proliferation void}
{self.add-application-property 'draw-updates true}
{self.add-application-property 'select-tolerance
                               CANVAS-MOUSE-TOLERANCE-DISTANCE}
{self.add-application-property 'rotate-from-corner false}
{self.add-application-property 'corner-rounding-amt 5}
{self.add-application-property 'corner-rounding-type 'corner}
{self.add-application-property 'add-point-previous true}
{self.add-application-property 'drag-by-outline true}
{self.add-application-property 'text-wrap-on false}
{self.add-application-property 'show-closed-polygon false}
{self.add-application-property 'select-by-enclosure true}
{self.add-application-property 'layers-showing #xFFFF}
{self.add-application-property 'rotate-selection-boxes true}
```

```
| and the canvas properties
{self.add-canvas-property 'line-color 'black}
{self.add-canvas-property 'line-width 1}
{self.add-canvas-property 'line-type 'solid}
{self.add-canvas-property 'arrow-type 0}
{self.add-canvas-property 'fill-pattern void}
{self.add-canvas-property 'fill-color 'blue}
{self.add-canvas-property 'font-justification 'left}

| add semaphores
{self.add-semaphore 'popup}
{self.add-semaphore 'mode}

| add sub-menu titles
{self.add-menu-heading void 'File before='Window}
{self.add-menu-heading void 'Edit after='File}
{self.add-menu-heading void 'Draw after='Edit}
{self.add-menu-heading void 'Modify after='Draw}
{self.add-menu-heading void 'Help after='Window}

| add "File" menu options
{self.add-menu-item
 {make-list 'File}
 'New
 action=new-file-menu-action
 }
{self.add-menu-item
 {make-list 'File}
 'Export
 action=export-menu-action
 }
{self.add-menu-item
 {make-list 'File}
 'Save
 action=save-menu-action
 }
{self.add-menu-item
 {make-list 'File}
 'Load
 action=load-menu-action
 }


{self.add-menu-item
 {make-list 'Edit}
 "Clear All"
```

```
 action=clear-all-menu-action}
{self.add-menu-item
 {make-list 'Edit}
 'Undo
 before="Clear All"
 action=undo-menu-action}
{self.add-menu-item
 {make-list 'Edit}
 'Redo
 after='Undo
 action=redo-menu-action}

{self.add-menu-heading {make-list 'Edit} 'Properties}
{self.add-menu-item
 {make-list 'Edit 'Properties}
 'Depth
 action=depth-dialog}
| help: about, tutorial

| add draw toolbar buttons
| line
{let tb:NewToolbarButton
  {set tb {new NewDrawModesToolbarButton
               {new LineModeEventHandler}
               label={new PolyLineModeButtonGraphic}
               value="Polyline"
               }}
  {self.add-toolbar-button 'modes 'draw tb}
  {self.add-menu-item
   {make-list 'Draw}
   'Polyline
   action={canvas-action
           e:PointerEvent={new CanvasPointerEvent}
           {tb.enter e}
           {tb.pointer-press e}
           {tb.leave e}
           }
   }
  } | let
| rect
{let tb:NewToolbarButton
  {set tb {new NewDrawModesToolbarButton
               {new RectangleModeEventHandler}
               label={new RectangleModeButtonGraphic}
               value="Rectangle"
               }}
  {self.add-toolbar-button 'modes 'draw tb}
```

```
  {self.add-menu-item
   {make-list 'Draw}
   'Rectangle
   action={canvas-action
           e:PointerEvent={new CanvasPointerEvent}
           {tb.enter e}
           {tb.pointer-press e}
           {tb.leave e}
           }
   }
  } | let


| add edit toolbar buttons
{let tb:NewToolbarButton
  {set tb {new NewEditModesToolbarButton
               {new SelectResizeEventHandler}
               label={new ResizeModeButtonGraphic}
               value="Resize"
               }}
  {self.add-toolbar-button 'modes 'draw tb}
  {self.add-menu-item
   {make-list 'Modify}
   'Select/Resize
   action={canvas-action
           e:PointerEvent={new CanvasPointerEvent}
           {tb.enter e}
           {tb.pointer-press e}
           {tb.leave e}
           }
   }
  } | let


{let tb:NewToolbarButton
  {set tb {new NewEditModesToolbarButton
               {new SelectRotateEventHandler}
               label={new RotateModeButtonGraphic}
               value="Rotate"
               }}
  {self.add-toolbar-button 'modes 'draw tb}
  {self.add-menu-item
   {make-list 'Modify}
   'Rotate
   action={canvas-action
           e:PointerEvent={new CanvasPointerEvent}
           {tb.enter e}
```

```
            {tb.pointer-press e}
            {tb.leave e}
            }
   }
  } | let

{let tb:NewToolbarButton
  {set tb {new NewEditModesToolbarButton
               {new SelectMoveEventHandler}
               label={new MoveModeButtonGraphic}
               value="Move"
               }}
  {self.add-toolbar-button 'modes 'draw tb}
  {self.add-menu-item
   {make-list 'Modify}
   'Move
   action={canvas-action
           e:PointerEvent={new CanvasPointerEvent}
           {tb.enter e}
           {tb.pointer-press e}
           {tb.leave e}
           }
   }
  } | let



| add property toolbar buttons
{let tb:NewToolbarButton
  {set tb {new NewPropertiesToolbarButton
               label={new LineWidthButtonGraphic}
               tooltip="Line Width"
               dialog-proc=line-width-dialog}}
  {self.add-toolbar-button 'properties 'canvassheet tb}
  {self.set-canvas-property-button 'line-width tb}
  }

{let tb:NewToolbarButton
  {set tb {new NewPropertiesToolbarButton
               label={new LineColorButtonGraphic}
               tooltip="Line Color"
               dialog-proc=line-color-dialog}}
  {self.add-toolbar-button 'properties 'canvassheet tb}
  {self.set-canvas-property-button 'line-color tb}
  }

{let tb:NewToolbarButton
```

```
    {set tb {new NewPropertiesToolbarButton
                 label={new FillColorButtonGraphic}
                 tooltip="Fill Color"
                 dialog-proc=fill-color-dialog}}
    {self.add-toolbar-button 'properties 'canvassheet tb}
    {self.set-canvas-property-button 'fill-color tb}
    }

  } | method: init
} | class: CanvasApp-v1-0
```

Bibliography

[1] Aldus Corporation (now Adobe). *SuperPaint*, 13 edition.

[2] Autodesk. *AutoCAD*, 13 edition.

[3] William Cheng. *tgif Source Code*, 3.0 edition.

[4] Curl Welcome Page.
URL http://curl.lcs.mit.edu/curl/curl/docs/overview.curl, 1998.

[5] Denaba Systems Incorporated. *Canvas*, 13 edition.

[6] Free Software Foundation. *XPaint Source Code*, 2.2 edition.

[7] FrontPage 98 – Overview.
URL http://www.microsof.com/products/prodref/571_ov.htm, 1998.

[8] M. Hostetter, D. Kranz, C. Seed, C. Terman, and S. Ward. The curl project.
URL http://curl.lcs.mit.edu/curl/, 1998.

[9] PowerPoint 97 for Windows 95 – Overview.
URL http://www.microsof.com/products/prodref/127_ov.htm, 1998.

[10] Presenting Adobe FrameMaker 5.5.
URL http://www.adobe.com/prodindex/framemaker/main.html, 1998.

[11] SoftQuad: Product Catalogue.
URL http://www.softquad.com/products/hotmetal/, 1998.

[12] Ivan Edward Sutherland. *Sketchpad: A Man-Machine Graphical Communication System.* M.I.T. Lincoln Laboratory, Lexington, Massachusetts, 1963.

[13] Visual Basic 5.0 Professional Edition – Overview.
URL http://www.microsof.com/products/prodref/195_ov.htm, 1998.

[14] Visual C++ 5.0 Professional Edition – Overview.
URL http://www.microsof.com/products/prodref/197_ov.htm, 1998.

[15] Visual J++ 1.1 – Overview.
URL http://www.microsof.com/products/prodref/221_ov.htm, 1998.

[16] X Consortium. *XFig Source Code*, 3.1 edition.