A Sample Garnet Program

Brad A. Myers

October 1993

Abstract

This file contains a sample program written using the Garnet Toolkit. The program is a simple graphical editor that allows the user to create boxes and arrows.

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

1. Introduction

The program in this file is implemented using the Garnet Toolkit, and is presented as an example of how to write programs using the toolkit. The program implements a graphical editor that allows the user to create boxes with textual labels which can be connected by lines. The lines have arrowheads, and go from the center of one box to the center of another. The boxes can be moved or changed size, and the arrows stay attached correctly. The boxes or lines can also be deleted, and the labels can be edited.

The sample program is in the file `demo-editor.lisp`, and a source and binary (compiled) version should be available in the `demos` sub-directory of the Garnet system files.

This graphical editor shows the use of:

- Constraints: to keep the arrows centered, to keep the name labels at the tops of boxes, etc.
- Opal objects: roundtangles, cursor-text, windows.
- Interactors: to choose which drawing mode (`menu-interactor`), to edit the text strings (two `text-interactors`), and to create new objects (`two-point-interactor`).
- Toolkit widgets: `Text-button-panel` (a form of menu), `graphics-selection` (to show which object is selected and allow it to be moved), and `arrow-lines`. These widgets have built in Opal objects and Interactors.
- Aggregadgets: to group the roundtangle and label string.
- Creating instances from prototypes (creating the new boxes and arrows).

This code is about 365 lines long, including comments, and took me two hours to code and one hour to debug. I did not use any higher-level Garnet tools to create it (it was all coded directly in Lisp).

2. Loading the Editor

After loading `Garnet-loader`, the following will load the editor:

```
(load (merge-pathnames "text-buttons-loader" Garnet-Gadgets-Pathname))
(load (merge-pathnames "graphics-loader" Garnet-Gadgets-Pathname))
(load (merge-pathnames "arrow-line-loader" Garnet-Gadgets-Pathname))

(load (merge-pathnames "demo-editor" Garnet-Demos-Pathname))
```

Under CMU CommonLisp, you can use the following instead:

```
(load "gadgets:text-buttons-loader")
(load "gadgets:graphics-loader")
(load "gadgets:arrow-line-loader")

(load "demos:demo-editor")
```

3. User Interface

A snap shot of the editor in use is shown in Figure 1.

The user interface is as follows. The menu at the top determines the current mode. When the roundtangle is outlined, the user can draw new boxes, and when the arrow is outlined, the user can draw new arrows. Press with either the left or right mouse buttons over one to change modes.

To create a new roundtangle, press with the *right* button in the workspace window (on the right) and hold

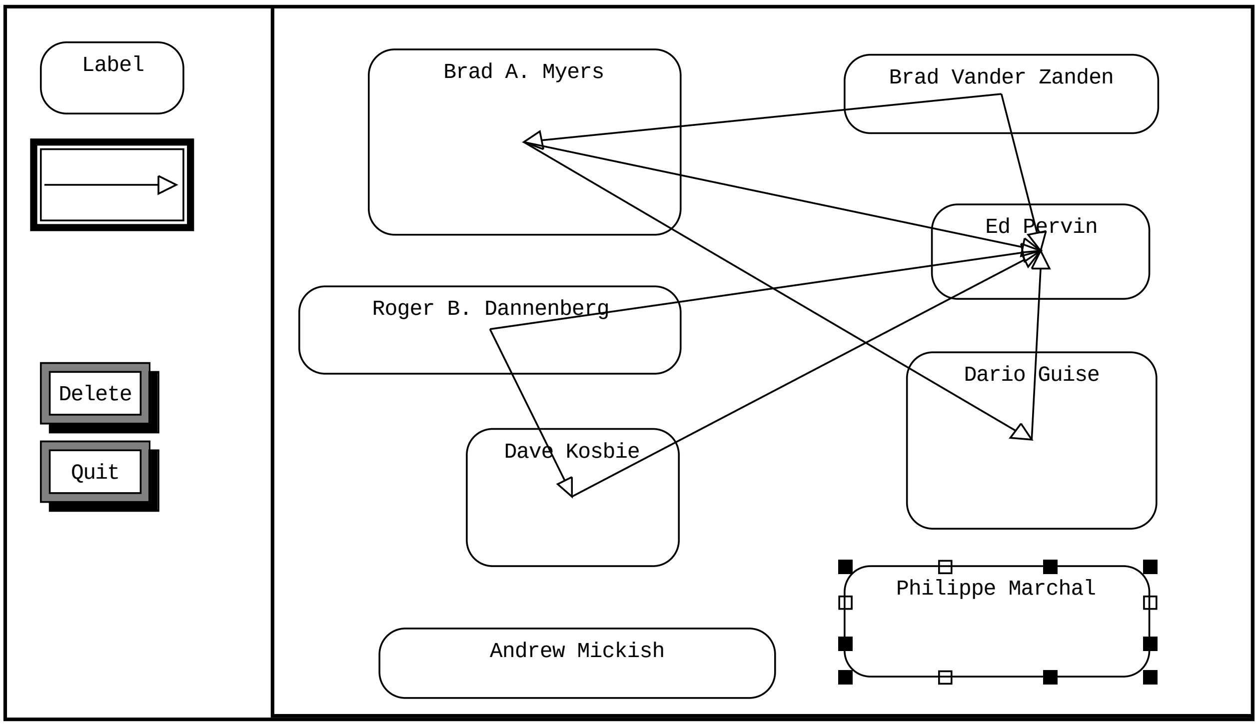

Figure 1: A Sample Garnet Application. The code for this application is listed at the end of this technical report.

down. Drag to the desired size and then release. Next, type in the new name. Various editing characters are supported during typing, including:

`^h, delete, backspace`: delete previous character.
`^d`: delete next character.
`^u`: delete entire string.
`^b, left-arrow`: go back one character.
`^f, right-arrow`: go forward one character.
`^a, home`: go to the beginning of the string.
`^e, end`: go to the end of the string.
`^y, insert`: insert the contents of the X cut buffer into the string at the current point.

When finished typing, press RETURN or any mouse button to stop.

To create a new arrow, when in arrow mode, press with the *right* button over a roundtangle and release over a different roundtangle. An arrow will be drawn starting at the center of the first roundtangle and going to the center of the other one.

Press with the *left* button on a roundtangle or an arrow to select it. Press on the background to cause there to be no selection. Press and release on the ‘‘Delete’’ button to delete the selected object.

If a roundtangle is selected, you can move it by pressing on a small white square with the left button and dragging to the new position and releasing. You can change its size by pressing with the left button on a black square. While the dotted outline box is displayed, you can abort the operation by moving outside the window and releasing, or by hitting ^G (control-G).

To change the string of a label, press on the label with the left button and begin typing. When finished typing, press RETURN or any mouse button to stop, or press ^G to abort the editing and return to the original string.

4. Overview of How the Code Works

The next section contains the actual code for the demo editor. This section presents some the parts of the design and serves as a guide to the code.

The standard ‘‘Garnet style’’ is to `USE-PACKAGE` the `KR` package, and directly reference all the other Garnet packages, so this is how the code is written. Functions such as `create-instance, s-value, o-formula, formula, gv,` and `gvl` are defined in `KR`.

The first part of the code creates *prototypes* of the basic items that the user will create: arrow lines and labeled boxes. When the editor is running, the code will create an *instance* of one of these prototypes to get a new set of objects to be displayed. The arrow line object is composed of one `arrow-line` from the Garnet-Gadget set, with some special *constraints* on its end-points. The arrow-line is parameterized by the two objects it is connected to. These two objects are kept in slots of the arrow-line: `:from-obj` and `:to-obj`. The constraints on the end-points of the arrow-line are expressed as *formulas* that cause the arrow to go from the center of the object stored in the `:from-obj` slot of the arrowline, to the center of the object in the `:to-obj` slot.

The labeled box is more complicated. It is composed of two parts: a rounded-rectangle (‘‘roundtangle’’) and a label. An *AggreGadget* is used to compose these together. The boundaries of the roundtangle are defined by the values in the `:box` slot, since the standard `Move-Grow-Interactor` modifies objects by setting this slot. The label string is constrained to be centered at the top of the roundtangle. The actual string used is stored both at the top level AggreGadget, and in the text object, so formulas are set up to keep these two `:string` slots having the same value.

The next function (`create-mode-menu`) creates the top menu that contains a label object and an arrow object. A feedback rectangle is also created to show what the current mode is. This feedback rectangle has formulas that keep it over whatever mode object is selected. An interactor is then created to allow the user to choose the mode.

The main command menu is created using `create-menu`, which simply creates an instance of a `garnet-gadget:text-button-panel` in the correct place. The functions to be executed are `delete-object` and `do-quit`, and these are defined next. The only trick here is that if a labeled box is deleted, the lines to it are also deleted. For quit, destroying a window automatically destroys all of its contents.

Creating a new object is fairly straightforward. The interactor is queried to find out whether to create a line or a box, and the appropriate kind of object is then created. Lines can only start or end in boxes, so the appropriate boxes are found. To appear in a window, the newly created objects must be added to an *aggregate* which is attached to the window, Here, the aggregate found by looking in the `:objs-aggregate` slot of the interactor.

Another important feature of the `Create-New-Obj` procedure is that if the object being created is a box, then it starts an interactor to allow the user to type the text label.

The top-level, exported procedure, `do-go`, starts everything up by creating a window, a sub-window to be the work area, and top-level aggregates for both windows. Another aggregate will hold the user-created objects. The `selection` object will show which object is selected, and also allow that object to be moved or grown (if it is not a line). Two text editing interactors are then created. One is used when a new object is created to have the user type in the initial name. This one is started explicitly using `Start-interactor` in `Create-New-Object`. The other interactor is used when the user presses on the text label of an object to edit the name.

Finally, the interactor to create new objects is defined. This one is a little complex, because it needs to decide whether to use a line or rectangle feedback based on the current mode.

The last step is to add the top level aggregates to the windows and call `update` to get the objects to appear. If you are running CMU CommonLisp, they will begin operating by themselves, but under other CommonLisps, the `Main-Event-Loop` call is needed to get the interactors to run. The `Exit-Main-Event-Loop` function in `Do-Quit` causes `Main-Event-Loop` to exit.

5. The Code

```
;;; -*- Mode: Lisp; Package: DEMO-EDITOR -*-
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;The Garnet User Interface Development Environment
;;;Copyright (c) 1989, Carnegie Mellon University
;;;All rights reserved.  The CMU software License Agreement specifies
;;;the terms and conditions for use and redistribution.
;;;
;;;If you want to use this code or anything developed as part of the
;;;Garnet project, please contact Brad Myers (Brad.Myers@CS.CMU.EDU).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;This file is a sample of a graphics editor created with Garnet.  It is
;;;designed to be a model for other code development, and therefore uses
;;;all the most up-to-date Garnet features.
;;;
;;;** Before loading this, you need to load:
;;;     * The Standard Garnet files
;;;     * text-buttons-loader from the gadgets
;;;     * graphics-loader from the gadgets
;;;     * arrow-line-loader from the gadgets
;;;
;;;** Call (demo-editor:Do-Go) to start and (demo-editor:Do-Stop) to stop **
;;;
;;;Designed and implemented by Brad A. Myers

(in-package "DEMO-EDITOR" :use '("KR" "LISP"))

(export '(Do-Go Do-Stop))

;;;
```

```
;;;--------------------------------------------------------------------------
;;;First create the prototypes for the box and lines
;;;--------------------------------------------------------------------------


(create-instance 'myarrowline garnet-gadgets:arrow-line
                 (:from-obj NIL) ;set this with the object this arrow is from
                 (:to-obj NIL)   ;set this with the object this arrow is from
                 (:x1 (o-formula (opal:gv-center-x (gvl :from-obj))))
                 (:y1 (o-formula (opal:gv-center-y (gvl :from-obj))))
                 (:x2 (o-formula (opal:gv-center-x (gvl :to-obj))))
                 (:y2 (o-formula (opal:gv-center-y (gvl :to-obj))))
                 (:open-p NIL)
                 (:visible (o-formula (and (gvl :from-obj)(gvl :to-obj))))
                 (:line-p T) ;so that the selection object will know what kind this is
                 )

(create-instance 'mylabeledbox opal:aggregadget
                 (:box (list 20 20 40 20)) ;this will be set by the
                                           ;interactors with the size of this box.

                 (:lines-at-this-box NIL)  ;Keep track of lines pointing
                                           ;to me, in case I am deleted.

                 ;;Set up a circular constraint between this string slot and the
                 ;;string slot in the label.  If either is changed, the other is
                 ;;automatically updated.  For circular constraints, it is
                 ;;important to have an initial value, here it is the empty string.
                 (:string (o-formula (gvl :label :string) ""))

                 (:line-p NIL) ;so that the selection object will know what kind this is
                 (:parts
                  `((:frame ,opal:roundtangle
                        (:radius 15)
                        (:left ,(o-formula (first (gvl :parent :box))))
                        (:top ,(o-formula (second (gvl :parent :box))))
                        (:width ,(o-formula (third (gvl :parent :box))))
                        (:height ,(o-formula (fourth (gvl :parent :box)))))
                    (:label ,opal:cursor-text
                        (:string ,(o-formula (gvl :parent :string) ""))
                        (:cursor-index NIL)
                        ;;center me horizontally with respect to the frame
                        (:left ,(o-formula
                                 (- (opal:gv-center-x (gvl :parent :frame))
                                    (floor (gvl :width) 2))))
                        (:top ,(o-formula
                                (+ (gvl :parent :frame :top) 5))))))))

;;;
```

```
;;;-----------------------------------------------------------------------
;;;Create main menu object
;;;-----------------------------------------------------------------------

;;Create an arrow and a box menu object, and put them in a menu, with an
;;interactor and feedback object to show which is selected.
;;Agg is the top level aggregate to put the menu in, and window is the window.
;;The :line-p slot of the agg is set with a formula to tell whether in line mode
;;or not.
(defun create-mode-menu (agg window)
  (let (feedback boxitem arrowitem)
    (setf boxitem (create-instance NIL mylabeledbox
                                   (:box (list 20 20 80 40))
                                   (:string "Label")))
    ;;the arrow will be inside a box.
    (setf arrowitem
          (create-instance NIL opal:aggregadget
                (:parts
                 `((:frame ,opal:rectangle
                      (:left 20)(:top 80)(:width 80)(:height 40))
                   (:line ,garnet-gadgets:arrow-line
                      (:open-p NIL)
                      (:x1 ,(o-formula (+ (gvl :parent :frame :left) 2)))
                      (:y1 ,(o-formula
                             (opal:gv-center-y (gvl :parent :frame))))
                      (:x2 ,(o-formula (+ (gvl :parent :frame :left) 76)))
                      (:y2 ,(o-formula (gvl :y1))))))))

    ;;The interactor (defined below) will set the :selected slot of the aggregate.
    ;;Use this to determine where the feedback should be.
    ;;We need to use formula rather than o-formula here so we can have a direct
    ;;reference to agg (use formula whenever you need to reference an object that
    ;;is not stored in a slot of the current object). Notice the use of
    ;;back-quote and comma to get a reference to the agg object.
    (setf feedback (create-instance NIL opal:rectangle
                          (:line-style opal:line-4)
                          (:filling-style NIL)
                          (:left (formula `(- (gv ',agg :selected :left) 6)))
                          (:top (formula `(- (gv ',agg :selected :top) 6)))
                          (:width (formula
                                   `(+ (gv ',agg :selected :width) 12)))
                          (:height (formula
                                    `(+ (gv ',agg :selected :height) 12)))
                          (:visible (formula `(gv ',agg :selected)))
                          (:draw-function :xor)
                          (:fast-redraw-p T)))
    (opal:add-components agg boxitem arrowitem feedback)

    ;;use the :menuobjs slot to hold the items that can be selected
    (s-value agg :menuobjs (list boxitem arrowitem))

    ;;default mode is the rectangle
    (s-value agg :selected boxitem)

    ;;The :line-p slot of the agg is set with a formula to tell whether in line mode or not.
    (s-value agg :line-p (formula `(eq (gvl :selected) ',arrowitem)))

    ;;now create an interactor to choose which mode
    (create-instance NIL inter:menu-interactor
                     (:window window)
                     (:start-event '(:leftdown :rightdown)) ;either one
                     (:start-where `(:list-element-of ,agg :menuobjs)))))


;;;
```

```
;;This creates the menu of commands.  For now, it only has "delete" and "quit" in it.
;;The menu is stored into the aggregate agg.  Returns the menu created.
(defun create-menu (agg)
  (let ((menu (create-instance NIL Garnet-gadgets:Text-Button-Panel
                        (:items '(("Delete" Delete-Object) ("Quit" Do-Quit)))
                        (:left 20)
                        (:top 200)
                        (:font Opal:Default-font)
                        (:shadow-offset 5)
                        (:final-feedback-p NIL))))
    (opal:add-components agg menu)
    menu))

;;;*******************************************************************
;;;Procedures to do the work
;;;*******************************************************************

;;;Delete-Line is called from delete object to delete lines
(defun Delete-Line(line-obj)
  (let ((from-obj (g-value line-obj :from-obj))
        (to-obj (g-value line-obj :to-obj)))
    ;;remove this line from the boxes' lists
    (s-value from-obj :lines-at-this-box
             (delete line-obj (g-value from-obj :lines-at-this-box)))
    (s-value to-obj :lines-at-this-box
             (delete line-obj (g-value to-obj :lines-at-this-box)))
    (opal:destroy line-obj)))

;;;Delete-object is called from the main menu routine
(defun Delete-Object (toolkit-obj menu-item)
  (declare (ignore menu-item))
  (let ((selected-obj (g-value toolkit-obj :selection-obj :value)))
    (if selected-obj
      (progn
        ;;first turn off selection
        (s-value (g-value toolkit-obj :selection-obj) :value NIL)
        ;;now delete object
        (if (g-value selected-obj :line-p)
            ;;then deleting a line
            (Delete-Line selected-obj)
            ;;else deleting a box
            (progn
              ;;first delete all lines to this box
              (dolist (line-at-box (g-value selected-obj :lines-at-this-box))
                (delete-line line-at-box))
              ;;now delete the box
              (opal:destroy selected-obj))))
        ;;else nothing selected
        (inter:beep))))

(defun Do-Quit (toolkit-obj menu-item)
  (declare (ignore menu-item))
  (opal:destroy (g-value toolkit-obj :window))
  ;;if not CMU CommonLisp, then exit the main event loop
  #-cmu (inter:exit-main-event-loop))

;;;
```

```
;;;Create a new object.  Get the type of object to create from the interactor.
;;;This procedure is called as the final-function of the two-point interactor.
(defun Create-New-Obj (inter point-list)
  (let ((agg (g-value inter :objs-aggregate))
        (line-p (g-value inter :line-p))) ;create a line or rectangle

    (if line-p
        ;;then create a line, first have to find the objects where the line is drawn
        (let ((from-box (opal:point-to-component agg (first point-list)
                                (second point-list) :type mylabeledbox))
              (to-box (opal:point-to-component agg (third point-list)
                                (fourth point-list) :type mylabeledbox))
              new-line)
          ;;If one end of the arrow is not inside a box, or is from and to the same box, then beep.
          (if (or (null from-box)(null to-box) (eq from-box to-box))
              (inter:beep)
              ;; else draw the arrow.
              (progn
                (setf new-line (create-instance NIL myarrowline
                                                  (:from-obj from-box)
                                                  (:to-obj to-box)))
                ;;keep track in case boxes are deleted so can delete this line.
                (push new-line (g-value from-box :lines-at-this-box))
                (push new-line (g-value to-box :lines-at-this-box))

                (opal:add-component agg new-line))))
        ;;else, create a new box
        (let ((textinter (g-value inter :textinter))
              (new-box (create-instance NIL mylabeledbox
                             (:box (copy-list point-list))))) ;have to make
                                                              ;a copy of list since
                                                              ;the interactor
                                                              ;re-uses the same list
          (opal:add-component agg new-box)
          ;;now start the interactor to allow the user to type the label.
          ;;Obj-to-change is the label object of the new box.
          (s-value textinter :obj-to-change (g-value new-box :label))
          (inter:start-interactor textinter)))))

;;;
```

```
;;;*******************************************************************
;;;Main procedures
;;;*******************************************************************

(defparameter current-window NIL) ;this global variable is only used for
                                  ; the debugging function below: do-stop

(defun Do-Go ()
  (let (top-win work-win top-agg work-agg selection objs-agg menu edit-text)
    ;;;create top-level window
    (setf top-win (create-instance NIL inter:interactor-window
                              (:left 10) (:top 10)
                     (:width 700) (:height 400)(:title "GARNET Sample Editor")
                     (:icon-title "Graphics Editor")))
    (setf current-window top-win)

    ;;;create window for the work area
    (setf work-win (create-instance NIL inter:interactor-window
                     (:left 150)
                     (:top -2) ;no extra border at the top
                     (:width (formula '(- (gv ',top-win :width) 150)))
                     (:height (formula '(gv ',top-win :height)))
                     (:border-width 2)
                     (:parent top-win)))

    ;;;create the top level aggregate in the windows
    (setq top-agg (create-instance NIL opal:aggregate
                     (:left 0)(:top 0)
                     (:width (formula '(gv ',top-win :width)))
                     (:height (formula '(gv ',top-win :height)))))

    (setq work-agg (create-instance NIL opal:aggregate
                     (:left 0)(:top 0)
                     (:width (formula '(gv ',work-win :width)))
                     (:height (formula '(gv ',work-win :height)))))
    ;;;create an aggregate to hold the user-created objects
    (setq objs-agg (create-instance NIL opal:aggregate
                     (:left 0)(:top 0)
                     (:width (formula '(gv ',work-win :width)))
                     (:height (formula '(gv ',work-win :height)))))
    (opal:add-component work-agg objs-agg)

    ;;;create menus
    (create-mode-menu top-agg top-win)
    (setf menu (create-menu top-agg))

    ;;;create a graphics selection object
    (setq selection (create-instance NIL Garnet-Gadgets:graphics-selection
                        (:start-where (list :element-of-or-none objs-agg))
                        (:movegrow-lines-p NIL) ;can't move lines
                        ;;move objects while cursor in the work window
                        (:running-where (list :in work-win))))
    (opal:add-component work-agg selection)

    ;;store the selection object in a new slot of the menu so that the delete
    ;;function can find which object is selected.
    (s-value menu :selection-obj selection)

    ;;;Create an interactor to edit the text of the labels when they are first
    ;;;created.  This interactor will never start by itself, but is started
    ;;;explicitly using Inter:Start-Interactor in the Create-New-Object function.
    (setf edit-text (create-instance NIL Inter:Text-Interactor
                       (:obj-to-change NIL) ;this is set when the interactor is started
                       (:start-event NIL) ;won't start by itself
                       (:start-where NIL) ;won't start by itself
                       (:stop-event '(#\return :any-mousedown)) ;either stops it
                       (:window work-win)))
    ;;cont., next page
```

```
    ;;The next interactor edits the text when the user presses on a string.
    (create-instance NIL Inter:Text-Interactor
                       (:stop-event '(#\return :any-mousedown)) ;either stops it
                       (:start-where (list :leaf-element-of objs-agg
                                           :type Opal:cursor-text))
                       ;;high priority so that if this one runs, the object
                       ;;underneath will not become selected.
                       (:waiting-priority inter:high-priority-level)
                       (:window work-win))

    ;;;create an interactor to create the new objects
    (create-instance NIL Inter:Two-Point-Interactor
                     (:start-event :rightdown)
                     (:start-where T)
                     (:running-where (list :in work-win))
                     (:window work-win)
                     (:abort-event '(#\control-G #\control-\g)) ;abort on ^G or ^g
                     (:line-p (formula `(gv ',top-agg :line-p)))
                     ;The next 2 slots are used by the Create-New-Obj procedure,
                     ;not by this interactor itself.
                     (:objs-aggregate objs-agg)
                     (:textinter edit-text)

                     (:feedback-obj
                      ;;use the feedback objects in the graphics-selection object
                      ;;pick which feedback depending on whether drawing line or box
                      (formula
                       `(if (gvl :line-p)
                            ',(g-value selection :line-movegrow-feedback)
                            ',(g-value selection :rect-movegrow-feedback))))
                     (:final-function #'Create-New-Obj))

    ;;;Now, add the aggregates to the window and update
    (s-value top-win :aggregate top-agg)
    (s-value work-win :aggregate work-agg)
    (opal:update top-win)  ;;will also update work-win

  ;;** Do-Go **
  (Format T "~%Demo-Editor:
  Press with left button on top menu to change modes (box or line).
  Press with left button on bottom menu to execute a command.
  Press with right button in work window to create a new object
        of the current mode.
  Boxes can be created anywhere, but lines must start and stop inside boxes.
  After creating a box, you should type the new label.
  Press with left button on text string to start editing that string.
        While editing a string, type RETURN or press a mouse button to stop.
  Press with left button in work window to select an object.
  Press with left button on white selection square to move an object.
  Press with left button on black selection square to change object size.
  While creating, moving, or growing a box, move outside window and release or
        hit ^G or ^g to abort.
  ~%")

    ;;if not CMU CommonLisp, then start the main event loop to look for events
    #-cmu (inter:main-event-loop)


    ;;return top window
    top-win))

;;** This is mainly for debugging, since usually the quit button in the menu will be used.
(defun Do-Stop ()
  (opal:destroy current-window))
```

Table of Contents