Demonstration Programs for Garnet

Brad A. Myers
Andrew Mickish

October 1993

Abstract

This file contains an overview of the demonstration programs distributed with the Garnet toolkit. These programs serve as examples of what Garnet can do, and also of how to write Garnet programs.

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

Probably the best way to learn about how to code using the Garnet Toolkit is to look at example programs. Therefore, we have provided a number of them with the Toolkit release. In addition, you can load and run the demos to see what kinds of things Garnet can do.

The ‘‘best’’ example program is `demo-editor`, which is included in this technical report. The other example programs serve mainly to show how particular special features of Garnet can be used.

Unfortunately, many of the demonstration programs were implemented before important parts of the Garnet Toolkit were implemented. For example, many of the demos do not use Aggregadgets and Aggrelists. These particular demos are *not* good examples of how we would code today. Hopefully, we will soon re-code all of these old demos using the newest features, but for the time being, you will probably only want to look at the code of the newer demos.

This document provides a guide to the demo programs, what they are supposed to show, and whether they are written with the latest style or not.

2. Loading and Compiling Demos

If for some reason the demos were not compiled during the standard installation procedure discussed in the Overview Manual, you can compile just the demos by executing (garnet-load "demos-src:demos-compiler"). This will generate new binaries for the demos, which will need to be copied from the `src/demos/` directory into your `bin/demos` directory.

Normally, the demonstration programs are *not* loaded by the standard Garnet loader. The best way to view the demos is to load the `garnet-loader` as usual and then load the Demos Controller:

```
(garnet-load "demos:demos-controller")
(demos-controller:do-go)
```

This will load the controller itself, but not any of the demos. It will display a window with a set of check buttons in it. Just click with the mouse on a button, and the corresponding demo will be loaded and started. Clicking on the check box again will stop the demo. Clicking again will restart it (but not re-load it). An instruction window will appear at the bottom of the screen with the instructions for the last demo started.

The `demos-controller` application features the `gg:mouseline` gadget. When you keep the mouse still over one of the x-buttons for about 2 seconds, a window will pop up with a short description of the corresponding demo. For more information about this gadget, see the appropriate section of the Gadgets Manual.

Using the `demos-controller` causes each demo file to be loaded as it is needed. If you wanted to load <u>all</u> of the demos at once (whether you eventually planned to use the `demos-controller` or not), you could set `user::load-demos-p` to be T before loading `garnet-loader`, or execute `load Garnet-Demos-Loader`.

All of the demos described here are in the sub-directory `demos`.

3. Running Demo Programs

To see a particular demo program, it is not necessary to use the Demos Controller described in section 2. Instead, the file can be loaded and executed by itself.

Almost all of the demonstration programs operate the same way. Once a file `demo-`*xxx* is loaded, it creates a package called `demo-`*xxx*. In this package are two procedures -- `do-go` to start the demo and `do-stop` to stop it. Therefore, to begin a demo of *xxx*, you would type: `(demo-`*xxx*`:do-go)`. The do-stop procedure destroys the window that the demo is running in. You can load and start as many demos as you like at the same time. Each will run in its own separate window.

The do-go procedure will print instructions in the Lisp window about how to operate the demonstration program.

Demos for the individual gadgets are all in the garnet-gadgets package and have unique names. Section 5.19 describes how to see these demos.

4. Double-Buffered Windows

All the demos can take advantage of the Opal feature for double-buffered windows. The do-go routine for each demo has an optional :double-buffered-p argument that defaults to NIL. For instance, to run demo-3d on a double-buffered window, say:

```
(demo-3d:do-go :double-buffered-p T)
```

and to run it normally, say:

```
(demo-3d:do-go)
```

5. Best Examples

5.1. GarnetDraw

There a useful utility called GarnetDraw which is a relatively simple drawing program written using Garnet. Since the file format for storing the created objects is simply a Lisp file which creates aggregadgets, you might be able to use GarnetDraw to prototype application objects (but Lapidary is probably better for this).

GarnetDraw uses many features of Garnet including gridding, PostScript printing, selection of all objects in a region, moving and growing of multiple objects, menubars, and the save-gadget and load-gadget dialog boxes. The editing functions like Cut, Copy, and Paste are implemented using the Standard-Edit module from garnet-gadgets, and objects can be cut and pasted between GarnetDraw and Gilt (since they share the same clipboard). Accelerators are defined for the menubar commands, like META-x for Cut and META-v for Paste.

GarnetDraw works like most Garnet programs: select in the palette with any button, draw in the main window with the right button, and select objects with the left button. Select multiple objects with shift-left or the middle mouse button. Change the size of objects by pressing on black handles and move them by pressing on white handles. The line style and color and filling color can be changed for the selected object and for further drawing by clicking on the icons at the bottom of the palette. You can also edit the shape of polylines: create a polyline, select it, and choose "Reshape" from the "Edit" menu.

5.2. Demo-Editor

Probably the best example program is the sample graphics editor in the file demo-editor.lisp. It demonstrates many of the basic components when building a Garnet application. This demo automatically loads and uses the text-button-panel, graphics-selection, and arrow-line gadgets.

5.3. Demo-Arith

`Demo-arith` is a simple visual programming interface for constructing arithmetic expressions. It uses constraints to solve the expressions. There are buttons for producing PostScript output from the picture. Also, you can create new objects using gestures by dragging with the middle mouse button (rather than selecting them from the palette). The instructions are printed when the program is started.

5.4. Demo-Grow

`Demo-grow` shows how to use the `graphics-selection` gadget. It uses the same techniques as in `demo-editor` (section 5.2).

5.5. Multifont and Multi-Line Text Input

`Demo-text` shows how multi-line, multi-font text input can be handled. It does not use Aggregadgets or any gadgets, but none are necessary.

5.6. Demo-Multifont

To see how to effectively use the multifont text object, along with its interactors, examine the `demo-multifont` demo. Most of the code is actually a good demonstration of how to use the `menubar` and `motif-scrolling-window-with-bars` gadgets, but the `multifont-text` objects and interactors are in there. Features demonstrated include word wrap and how to changing the fonts with the special multifont accelerators.

The `lisp-mode` feature of `multifont-text` is also shown in this demo. Select "Toggle Lisp Mode" from the "Edit" menu, and type in a lisp expression (like a `defun` definition). As you hit return, the next line will be automatically indented according to standard lisp conventions. Hitting the tab key will re-indent the current line.

5.7. Creating New Objects

`Demo-twop` shows how new lines and new rectangles can be input. It uses the same techniques as in `demo-editor` (section 5.2).

5.8. Angles

There are two programs that demonstrate how to use the angle interactor. `Demo-angle` contains circular gauges (but see the `gauge` gadget—section 5.19), as well as a demonstration of how to use the ‘‘angle-increment’’ parameter to the angle `:running-action` procedure.

`Demo-clock` shows a clock face with hands that can be rotated with the mouse.

5.9. Aggregraphs

The `demo-graph` file is an example of many features of Aggregraphs.

5.10. Scroll Bars

Although sliders and scroll bars are provided in the Garnet Gadget set (the `gadgets` subdirectory), the file `demo-scrollbar` contains some alternative scroll bar objects. The Macintosh scroll bar in this demo was written in the old Garnet style, but there are new versions of scroll bars in the OpenLook, Next, and Motif style.

To see the demo of all four scroll bars, use the functions `demo-scrollbar:do-go` and `demo-scrollbar:do-stop` as usual. There are also functions that display the scroll bars individually called `mac-go`, `open-go`, `next-go`, and `motif-go`.

5.11. Menus

`Demo-menu` shows a number of different kinds of menus that can be created using Garnet. All of them were implemented using Aggregadgets and Aggrelists.

5.12. Animation

`Demo-animator` uses background animation processes to move several objects in a window. One of the objects is a walking figure which moves across the screen by rapidly redrawing a pixmap.

`Demo-fade` shows a simple animation for the Garnet acronym.

`Demo-logo` performs the same animation as `demo-fade`, but it also includes the Garnet logo.

5.13. Garnet-Calculator

The `garnet-calculator` has the look and feel of `xcalc`, the calculator supplied by X windows, but it is more robust. The calculator is a self-contained tool, and can be integrated inside a larger Garnet application.

You can load the demo with `(garnet-load "demos:garnet-calculator")`. To run it, execute `(garnet-calculator:do-go)`.

`garnet-calculator:Start-Calc &key` *double-buffered-p* [*Function*]

`garnet-calculator:Stop-Calc` *app-object* `&optional (`*destroy-app-object?* `T)` [*Function*]

The function `start-calc` creates and returns a calculator "application object" that can be used by a larger interface, and this object should be passed as the *app-object* parameter to `stop-calc`.

5.14. Browsers

The files `demo-schema-browser` and `demo-file-browser` show two uses of the `browser-gadget`.

5.15. Demo-Virtual-Agg

To show off an example of virtual-aggregates, load Demo-Virtual-Agg and say:

```
(demo-virtual-agg:do-go :num-dots 1000)
```

`Demo-virtual-agg:do-go` takes a single optional keyed parameter `:num-dots` which tells how many circles should appear in a window. The default is 1000.

The first 1000 circles are read in from circles.data in the `user::Garnet-DataFile-PathName` directory (because that's faster) and the rest are chosen randomly. A '.' is printed out for every ten circles.

You will also see a little star in the upper left on the screen, in front of the `virtual-aggregate`, and a big gray rectangle underneath the `virtual-aggregate`. These are just to show that the update algorithm is working reasonably well.

Clicking with the left button creates a new circle (of random radius and color) where you clicked.

Clicking with the right button "destroys" the top-most circle underneath where you clicked, or beeps if there was nothing under there.

Clicking on the little star and dragging moves the little star.

Clicking shift-middle causes the circle underneath the cursor to change to a different random color. (This shows off `change-item`.)

Clicking shift-right causes the entire `virtual-aggregate` to disappear or reappear.

5.16. Demo-Pixmap

This new demo shows a two-dimensional `virtual-aggregate` in action. Here, the `virtual-aggregate` is a 50 X 50 array of 5 X 5 rectangles. Each rectangle can be colored from the color palette, and the pattern of colored rectangles is reflected in a pixmap.

You can load a pixmap into the demo (e.g., from the directory `Garnet-Pixmap-PathName`), edit the pixmap with the color palette and virtual-aggregate, and then save the pixmap to a new file. You can also generate PostScript files from this demo, though you have to have a Level 2 printer (that defines the PostScript function `colorimage`) to print a color pixmap image.

5.17. Demo-Gesture

`Demo-gesture` is an example of how the new gesture-interactor can be used in an interface. In this demo, you can create perfect circles and rectangles by drawing rough approximations with the mouse, which are interpreted by the gesture recognizer. Gestures may also be used to copy and delete the shapes you have created.

5.18. Demo-Unidraw

`Demo-Unidraw` is a gesture-based text editor, which allows you to enter characters with freehand drawing using the mouse. The gestures that this demo understands are comprised of a shorthand alphabet devised by David Goldberg at Xerox Parc. The gesture patterns are shown in the middle of the demo window, and the canvas for drawing gestures is at the bottom. As the demo recognizes the gestures you draw, it selects the corresponding gesture and puts the new character in the text window.

5.19. Gadget Demos

There are separate demo programs of some of the gadgets in the files `demo-gadgets` and `demo-motif`. Each of these packages export the usual `do-go` and `do-stop` procedures, and can be found in the `demos` directory.

Other good examples are the Garnet Gadgets, stored in the `gadgets` sub-directory. These were *all* written using the latest Garnet features. At the end of almost all gadget files is a small demo program showing how to use that gadget. Since all the gadgets are in the same package (`garnet-gadgets`), the gadget demos all have different names. They are:

- `Arrow-line-go, Arrow-line-stop` - to demonstrate arrow-lines
- `Error-gadget-go, Error-gadget-stop` - to demonstrate both the error gadget and the query gadget
- `Gauge-go, Gauge-stop` - to demonstrate circular gauges
- `H-scroll-go, H-scroll-stop` - to demonstrate standard horizontal scroll bars
- `H-slider-go, H-slider-stop` - to demonstrate standard horizontal sliders
- `Labeled-box-go, Labeled-box-stop` - to demonstrate labeled text-type-in objects
- `Menu-go, Menu-stop` - to demonstrate a standard menu
- `Menubar-go, Menubar-stop` - to demonstrate pull-down menus
- `Motif-Check-Buttons-go, Motif-Check-Buttons-stop` - to demonstrate Motif

style check buttons
- `Motif-Error-Gadget-go, Motif-Error-Gadget-stop` - to demonstrate both the motif error gadget and the motif query gadget
- `Motif-Gauge-go, Motif-Gauge-stop` - to demonstrate the Motif style gauge
- `Motif-H-Scroll-go, Motif-H-Scroll-stop` - to demonstrate Motif style horizontal scroll bars
- `Motif-Menu-go, Motif-Menu-stop` - to demonstrate the Motif style menus
- `Motif-Menubar-go, Motif-Menubar-stop` - to demonstrate the Motif style menubar, with accelerators
- `Motif-Option-Button-go, Motif-Option-Button-stop` - to demonstrate the Motif style version of this popup menu gadget, whose button changes labels according to the menu selection
- `Motif-Radio-Buttons-go, Motif-Radio-Buttons-stop` - to demonstrate Motif style radio buttons
- `Motif-Scrolling-Labeled-Box-go, Motif-Scrolling-Labeled-Box-stop` - to demonstrate the Motif style text-type-in field
- `Motif-Scrolling-Window-With-Bars-go, Motif-Scrolling-Window-With-Bars-stop` - to demonstrate the Motif style scrolling window gadget
- `Motif-Slider-go, Motif-Slider-stop` - to demonstrate the vertical Motif slider
- `Motif-Text-Buttons-go, Motif-Text-Buttons-stop` - to demonstrate Motif style text buttons
- `Motif-Trill-go, Motif-Trill-stop` - to demonstrate the Motif style trill device
- `Motif-V-Scroll-go, Motif-V-Scroll-stop` - to demonstrate the Motif vertical scroll bar
- `Mouseline-go, Mouseline-stop` - to demonstrate the mouseline and "balloon help" string
- `Multifont-Gadget-go, Multifont-Gadget-stop` - to demonstrate the gadget which is a conglomeration of a multifont-text, a focus-multifont-textinter, and a selection-interactor
- `Option-Button-go, Option-Button-stop` - to demonstrate this kind of popup menu gadget, whose button label changes according to the menu selection
- `Popup-Menu-Button-go, Popup-Menu-Button-stop` - to demonstrate this kind of popup menu gadget, whose button label is fixed and may be a bitmap or other object
- `Prop-Sheet-For-Obj-go, Prop-Sheet-For-Obj-stop` - to demonstrate how prop-sheets can be used to change slot values of Garnet objects
- `Radio-Buttons-go, Radio-Buttons-stop` - to demonstrate radio buttons
- `Scrolling-Input-String-go, Scrolling-Input-String-stop` - to demonstrate the scrolling input string gadget
- `Scrolling-Labeled-Box-go, Scrolling-Labeled-Box-stop` - to demonstrate the standard scrolling labeled box
- `Scrolling-Menu-go, Scrolling-Menu-stop` - to demonstrate the scrolling menu gadget
- `Scrolling-Window-go, Scrolling-Window-stop` - to demonstrate the standard scrolling window
- `Scrolling-Window-With-Bars-go, Scrolling-Window-With-Bars-stop` - to demonstrate the scrolling window with attached vertical and horizontal scroll bars
- `Text-Buttons-go, Text-Buttons-stop` - to demonstrate buttons with labels inside
- `Trill-go, Trill-stop` - to demonstrate the trill-device gadget
- `V-scroll-go, V-scroll-stop` - to demonstrate standard vertical scroll bars
- `V-slider-go, V-slider-stop` - to demonstrate standard vertical sliders
- `X-Buttons-go, X-Buttons-stop` - to demonstrate X buttons

Each of these has its own loader file, named something like *xxx*`-loader` for gadget *xxx*. See the Gadgets manual for a table of loader file names.

5.20. Real-Time Constraints and Performance

The program `demo-manyobjs` was written as a test of how fast the system can evaluate constraints. The `do-go` procedure takes an optional parameter of how many boxes to create. Each box is composed of four Opal objects.

6. Old Demos

6.1. Moving and Growing Objects

The best example of moving and growing objects is `demo-grow` (section 5.4).

In addition, `demo-moveline` shows how the `move-grow-interactor` can be used to move either end of a line.

6.2. Menus

`Demo-3d` shows some menus and buttons where the item itself moves when the user presses over it, in order to simulate a floating button.

7. Demos of Advanced Features

7.1. Using Multiple Windows

`Demo-multiwin` shows how an interactor can be used to move objects from one window to another. For more information, see the Interactors manual.

7.2. Modes

`Demo-mode` shows how you can use the `:active` slot of an interactor to implement different modes. For more information, see the Interactors manual.

7.3. Using Start-Interactor

`Demo-sequence` shows how to use the `inter:start-interactor` function to have one interactor start another interactor without waiting for the second one's start event. Another example of the use of `inter:start-interactor` is in `demo-editor` (section 5.2) to start editing the text label after drawing a box. For more information on `start-interactor`, see the Interactors manual.

Index

Table of Contents