# The Shape of PSIBER Space: PostScript Interactive Bug Eradication Routines

*Don Hopkins*

don@brillig.umd.edu
University of Maryland
Human-Computer Interaction Lab
Computer Science Department
College Park, Maryland 20742

*ABSTRACT*

The PSIBER Space Deck is an interactive visual user interface to a graphical programming environment, the NeWS window system. It lets you display, manipulate, and navigate the data structures, programs, and processes living in the virtual memory space of NeWS. It is useful as a debugging tool, and as a hands on way to learn about programming in PostScript and NeWS.

## 1. INTRODUCTION

‘‘Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts ... A graphic representation of data abstracted from the banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding ....’’

[Gibson, Neuromancer]

The PSIBER Space Deck is a programming tool that lets you graphically display, manipulate, and navigate the many PostScript data structures, programs, and processes living in the virtual memory space of NeWS.

The Network extensible Window System (NeWS) is a multitasking object oriented PostScript programming environment. NeWS programs and data structures make up the window system kernel, the user interface toolkit, and even entire applications.

The PSIBER Space Deck is one such application, written entirely in PostScript, the result of an experiment in using a graphical programming environment to construct an interactive visual user interface to itself.

It displays views of structured data objects in overlapping windows that can be moved around on the screen, and manipulated with the mouse: you can copy and paste data structures from place to place, execute them, edit them, open up compound objects to see their internal structure, adjust the scale to shrink or magnify parts of the display, and pop up menus of other useful commands. Deep or complex data structures can be more easily grasped by applying various views to them.

There is a text window onto a NeWS process, a PostScript interpreter with which you can interact (as with an ‘‘executive’’). PostScript is a stack based language, so the window has a spike sticking up out of it, representing the process’s operand stack. Objects on the process’s stack are displayed in windows with their tabs pinned on the spike. (See figure 1) You can feed PostScript expressions to the interpreter by typing them with the keyboard, or pointing and clicking at them with the mouse, and the stack display will be dynamically updated to show the results.

Not only can you examine and manipulate the objects on the stack, but you can also manipulate the stack directly with the mouse. You can drag the objects up and down the spike to change their order on the stack, and drag them on and off the spike to push and pop them; you can take objects off the spike and set them aside to refer to later, or close them into icons so they don't take up as much screen space.

NeWS processes running in the same window server can be debugged using the existing NeWS debug commands in harmony with the graphical stack and object display.

The PSIBER Space Deck can be used as a hands on way to learn about programming in PostScript and NeWS. You can try out examples from cookbooks and manuals, and explore and enrich your understanding of the environment with the help of the interactive data structure display.

## 2. INTERACTING WITH DATA

A PostScript object is a reference to any piece of data, that you can push onto the stack. (The word ''object'' is used here in a more general sense than in ''object oriented programming.'' The words ''class'' and ''instance'' are used for those concepts.) Each object has a type, some attributes, and a value. PostScript objects are dynamically typed, like Lisp objects, not statically typed, like C variables. Each object is either literal or executable. This attribute effects whether the interpreter treats it as data or instructions. Compound objects, such as arrays and dictionaries, can contain references to other objects of any type. [Adobe, Red, Green, and Blue books] [Sun, NeWS 1.1 Manual] [Gosling, The NeWS Book]

### 2.1. Viewing Data

Objects on the PSIBER Space Deck appear in overlapping windows, with labeled tabs sticking out of them. Each object has a label, denoting its type and value, i.e. ''integer 42''. Each window tab shows the type of the object directly contained in the window. Objects nested within other objects have their type displayed to the left of their value. The labels of executable objects are displayed in italics.

#### 2.1.1. Simple Objects

Figure 1 shows some simple objects: an integer, a boolean, a literal string, an executable string, a literal name, and an executable name -- the results of executing the string ''42 false (flamingo) (45 cos 60 mul) cvx /foobar /executive cvx''.

Strings are delimited by parenthesis: ''string (flamingo)''. Literal names are displayed with a slash before them: ''name /foobar''. Executable names are displayed in italics, without a leading slash: *''name executive''*. Names are most often used as keys in dictionaries, associated with the values of variables and procedures.

#### 2.1.2. Compound Objects

Compound objects, which contain other objects, can be displayed closed or opened. The two basic kinds of compound objects are arrays and dictionaries. Arrays are indexed by integers, and dictionaries are indexed by keys.

Figure 2 shows a literal array, an executable array, and a dictionary.

An opened compound object is drawn with lines fanning out to the right, leading from the object to its elements, which are labeled as ‘‘index: type value’’, in a smaller point size.

Literal arrays are displayed with their length enclosed in square brackets: ‘‘array [6]’’. Executable arrays (procedures) are displayed in italics, with their length enclosed in braces: *‘‘array {37}’’*. The lines fanning out from an opened array to its elements are graphically embraced, so they resemble the square brackets or braces in the label of a literal or executable array.

PostScript arrays are polymorphic: Each array element can be an object of any type. A PostScript procedure is just an executable array of other objects, to be interpreted one after the other.

The label of a dictionary shows the number of keys it contains, a slash, and the maximum size of the dictionary, enclosed in angled brackets: ‘‘dict <5/10>’’.

The lines that fan out from opened dictionaries resemble the angled brackets appearing in their labels.

Dictionaries associate keys with values. The key (an index into a dictionary) can be an object of any type (except null), but is usually a name. The value can be anything at all. Dictionaries are used to hold the values of variables and function definitions, and as local frames, structures, lookup tables, classes, instances, and lots of other things -- they’re very useful!

The dictionary stack defines the scope of a PostScript process. Whenever a name is executed or loaded, the dictionaries on the dictionary stack are searched, in top to bottom order.

### 2.1.3. Classes, Instances, and Magic Dictionaries

NeWS uses an object oriented PostScript programming package, which represents classes and instances with dictionaries. [Densmore, Object Oriented Programming in NeWS] [Gosling, The NeWS Book]

When a class dictionary is displayed, the class name is shown, instead of the ‘‘dict’’ type: ‘‘Object <10/200>’’. When an instance dictionary is displayed, its type is shown as a period followed by its class name: ‘‘.SoftMenu <31/200>’’.

Figure 3 shows the class dictionary of Object, and the instance dictionary of the NeWS root menu.

Magic dictionaries are certain types of NeWS objects, such as processes, canvases, and events, that act as if they were dictionaries, but have some special internal representation. They have a fixed set of keys with special meanings (such as a process’s ‘‘/OperandStack’’, or a canvas’s ‘‘/TopChild’’), that can be accessed with normal dictionary operations. Special things may happen when you read or write the values of the keys (for example, setting the ‘‘/Mapped’’ key of a canvas to false makes it immediately disappear from the screen). [Sun, NeWS 1.1 Manual] [Gosling, The NeWS Book]

Figure 4 shows a canvas magic dictionary (the framebuffer), and a process magic dictionary, with some interesting keys opened.

### 2.1.4. View Characteristics

The views of the objects can be adjusted in various ways. The point size can be changed, and well as the shrink factor by which the point size is multiplied as the structure gets deeper. The point size is not allowed to shrink smaller than 1, so that labels will never have zero area, and it will always be possible to select them with the mouse. If the shrink factor is greater than 1.0, the point size increases with depth.

The nested elements of a compound object can be drawn either to the right of the object label, or indented below it. When the elements are drawn indented below the label, it is not as visually obvious which elements are nested inside of which object, but it takes up a lot less screen space than drawing the elements to the right of the label does.

Any of the view characteristics can be set for a whole window, or for any nested object and its children within that window.

Figure 5 shows some nested structures with various point sizes and shrink factors, with elements opened to the right and below.

## 2.2. Editing Data

There are many ways to edit the objects displayed on the screen. There are functions on menus for converting from type to type, and for changing the object's attributes and value. You can replace an element of a compound object by selecting another object, and pasting it into the element's slot. There are also a bunch of array manipulation functions, for appending together arrays, breaking them apart, rearranging them, and using them as stacks. You must be careful which objects you edit, because if you accidentally scramble a crucial system function, the whole window system could crash.

## 2.3. Peripheral Controls

Peripheral controls are associated views that you can attach to an object, which are not directly contained within that object. They are visually distinct from the elements of a compound object, and can be used to attach editor buttons, computed views, and related objects. Several useful peripheral views have been implemented for manipulating various data types.

There are three types of numeric editors: the step editor, the shift editor, and the digit editor. The step editor has ‘‘++’’ and ‘‘--’’ buttons to increment and decrement the number it's attached to, by the parameter ‘‘Step’’. The shift editor has ‘‘**’’ and ‘‘//’’ buttons to multiply and divide the number it's attached to, by the parameter ‘‘Shift’’. The ‘‘Step’’ and ‘‘Shift’’ parameters appear in the peripheral views as normal editable numbers, to which you can attach other numeric editors, nested as deep as you like. The digit editor behaves like a numeric keypad, with buttons for the digits 0 through 9, ‘‘Rubout’’, ‘‘Clear’’, and ‘‘+-’’.

The boolean editor has ‘‘True’’, ‘‘False’’, and ‘‘Not’’ buttons that do the obvious things, and a ‘‘Random’’ button, that sets the boolean value randomly. Since the button functions are just normal data, you can open up the ‘‘Random’’ button and edit the probability embedded in the function ‘‘{random 0.5 lt}’’.

You can open up a definition editor on a name, to get editable references to every definition of the name on the dictionary stack (or in the context to which the enclosing class editor is attached).

The scroller editor allows you to view a reasonably sized part of a large array or dictionary. The peripheral controls include a status line telling the size of the object and how much of it is shown in the view, ‘‘Back’’ and ‘‘Next’’ buttons for scrolling the view, and a ‘‘Size’’ parameter that controls the number of elements in the view. You can edit the ‘‘Size’’ parameter of the scrolling view by attaching a numeric editor to it, or dropping another number into its slot, and it will take effect next time you scroll the view.

When you open a class editor, it attaches the following peripheral views: ‘‘ClassDicts’’, an array of the class dictionaries, ‘‘SubClasses’’, an array of a class's subclasses, ‘‘InstanceVars’’, an array of instance variable names, ‘‘ClassVars’’, an array class variable names, and ‘‘Methods’’, an array of method names. You can open up scrolling views on the arrays, and open up definition editors on the names, and you will be able to examine and edit the definitions in the class.

The canvas editor gives you a graphical view of the canvas's relation to its parent, and an array of the canvas's children. You can grab the graphical view of the canvas with the mouse and move the canvas itself around. You can open up the array of child canvases (with a scroller editor if you like), and attach canvas views to them, too.

Figure 6 shows some digit editors, step editors, shift editors, a boolean editor, and a canvas editor. Figure 7 shows a class editor, some scroller editors, name editors, and digit editors.

## 2.4. Printing Distilled PostScript

The data structure displays (including those of the Pseudo Scientific Visualizer, described below) can be printed on a PostScript printer by capturing the drawing commands in a file.

Glenn Reid's ''Distillery'' program is a PostScript optimizer, that executes a page description, and (in most cases) produces another smaller, more efficient PostScript program, that prints the same image. [Reid, The Distillery] The trick is to redefine the path consuming operators, like fill, stroke, and show, so they write out the path in device space, and incremental changes to the graphics state. Even though the program that computes the display may be quite complicated, the distilled graphical output is very simple and low level, with all the loops unrolled.

The NeWS distillery uses the same basic technique as Glenn Reid's Distillery, but it is much simpler, does not optimize as much, and is not as complete.

## 3. INTERACTING WITH THE INTERPRETER

In PostScript, as in Lisp, instructions and data are made out of the same stuff. One of the many interesting implications is that tools for manipulating data structures can be used on programs as well.

### 3.1. Using the Keyboard

You can type PostScript expressions into a scrolling text window, and interact with the traditional PostScript ''executive,'' as you can with ''psh'' to NeWS or ''tip'' to a laser printer. Certain function keys and control characters do things immediately when you type them, such as input editing, selecting the input, pushing or executing the selection, and completing names over the dictionary stack (like ''tcsh'' file name completion).

### 3.2. Using the Mouse

The mouse can be used to select data, push it on the stack, execute it, and manipulate it in many ways.

Pointing the cursor at an object and clicking the ''Menu'' button pops up a menu of operations that can be performed on it. All data types have the same top level pop-up menu (for uniformity), with a type specific submenu (for diversity). There are lots of commands for manipulating the object and the view available via pop-up menus.

You can select any object by clicking the ''Point'' button on it. A printed representation of the current selection is always displayed in a field at the top of the scrolling text window. If you click the Point button over an object whose label is too small to read, it will appear in the selection field, in a comfortable font.

Each object has its own button handler function that is called when you click the ''Adjust'' button on it. The default ''Adjust'' handler implements ''drag'n'dropping''. If you drop an object onto itself, its view toggles open or closed. If you drop it on top of a compound object element, it is stored into that memory location. If you drop it over an unoccupied spot, a new window viewing the object appears on the deck.

Another useful ‘‘Adjust’’ handler simply executes the object that was clicked on. This can be used to make buttons out of executable names, arrays, and strings.

### 3.3. Using Dictionaries as Command Pallets

A PostScript dictionary can be used as a pallet of commands, by defining a bunch of useful functions in a dictionary, opening it up, and executing the functions with the mouse. You can open up the functions to see their instructions, and even edit them!

### 3.4. Using a Text Editor

It is very helpful to be running a text editor on the source code of a PostScript program, while you are debugging it. You can select chunks of source from the text editor, and execute them in the PSIBER Space Deck (in the appropriate context). This is especially useful for redefining functions of a running program in memory, as bugs are discovered and fixed in the source code. It saves you from having to kill and restart your application every time you find a trivial bug.

## 4. DEBUGGING PROGRAMS

The NeWS debugger lets you take on and examine the state of a broken process. [Sun, NeWS 1.1 Manual] [Gosling, The NeWS Book] The debugger is a PostScript program originally written for use with ‘‘psh’’ from a terminal emulator. It is notorious for being difficult to use, but quite powerful. However, the debugger is much nicer in conjunction with the graphical stack, the object display, and a pallet of handy debugging commands, that you can invoke with the mouse.

When you enter a broken process with the debugger, you get a copy of its operand and dictionary stacks. You can examine and manipulate the objects on the stack, look at the definitions of functions and variables, and execute instructions in the scope of the broken process. You can change the stack, copy it back, and continue the process, or just kill it. You can push onto the spike the broken process, its dictionary stack, and its execution stack, and open them up to examine the process’s state.

I use the debugger extensively in developing the PSIBER Space Deck, both from a terminal emulator and from the deck itself. Using the deck to debug itself is an interesting experience. One of the most common uses is to redefine a function by selecting some text in from Emacs and executing it. But it’s still easy to make a mistake and crash it.

## 5. THE USER INTERFACE

### 5.1. Pie Menus

The mouse button functions and menu layouts were designed to facilitate gestural interaction, to simulate the feel of tweaking and poking at real live data structures.

There are several ‘‘pull out’’ pie menus, that use the cursor distance from the menu center as an argument to the selection.

The pie menu that pops up over data objects has the commonly used functions ‘‘Push,’’ ‘‘Exec,’’ and ‘‘Paste’’ positioned in easily selected directions (up, down, and left). Once you are familiar enough with the interface to ‘‘mouse ahead’’ into the menus, with quick strokes of the mouse in the appropriate direction, interaction can be very swift. [Callahan, A Comparative Analysis of Pie Menu Performance] [Hopkins, A Pie Menu Cookbook]

When you mouse ahead through a pie menu selection quickly enough, the menu is not displayed, and the shape of a pac-man briefly flashes on the screen, with its mouth pointing in the direction of the selected menu item. This ‘‘mouse ahead display suppression’’ speeds up interaction considerably by avoiding unnecessary menu display, and makes it practically impossible for the casual observer to follow what is going on. The flashing pac-man effect gives you some computationally inexpensive feedback of the menu selection, and reassures observers that you are racking up lots of points.

### 5.2. Tab Windows

The objects on the deck are displayed in windows with labeled tabs sticking out of them, showing the data type of the object. You can move an object around by grabbing its tab with the mouse and dragging it. You can perform direct stack manipulation, pushing it onto stack by dragging its tab onto the spike, and changing its place on the stack by dragging it up and down the spike. It implements a mutant form of ‘‘Snap-dragging’’, that constrains non-vertical movement when an object is snapped onto the stack, but allows you to pop it off by pulling it far enough away or lifting it off the top. [Bier, Snap-dragging] The menu that pops up over the tab lets you do things to the whole window, like changing view characteristics, moving the tab around, repainting or recomputing the layout, and printing the view.

## 6. THE METACIRCULAR POSTSCRIPT INTERPRETER

A program that interprets the language it is written in is said to be ‘‘metacircular’’. [Abelson, Structure and Interpretation of Computer Programs] Since PostScript, like Scheme, is a simple yet powerful language, with procedures as first class data structures, implementing ‘‘ps.ps’’, a metacircular PostScript interpreter, turned out to be straightforward (or drawrofthgiarts, with respect to the syntax). A metacircular PostScript interpreter should be compatible with the ‘‘exec’’ operator (modulo bugs and limitations). Some of the key ideas came from Crispin Goswell’s PostScript implementation. [Goswell, An Implementation of PostScript]

The metacircular interpreter can be used as a debugging tool, to trace and single step through the execution of PostScript instructions. It calls a trace function before each instruction, that you can redefine to trace the execution in any way. One useful trace function animates the graphical stack on the PSIBER Space Deck step by step.

The meta-execution stack is a PostScript array, into which the metacircular interpreter pushes continuations for control structures. (forall, loop, stopped, etc...) A continuation is represented as a dictionary in which the state needed by the control structure is stored (plus some other information to help with debugging).

It is written in such a way that it can interpret itself: It has its own meta-execution stack to store the program’s state, and it stashes its own state on the execution stack of the interpreter that’s interpreting it, so the meta-interpreter’s state does not get in the way of the program it’s interpreting.

It is possible to experiment with modifications and extensions to PostScript, by revectoring functions and operators, and modifying the metacircular interpreter.

The metacircular interpreter can serve as a basis for PostScript algorithm animation. One very simple animation is a two dimensional plot of the operand stack depth (x), against the execution stack depth (y), over time.

## 7. THE PSEUDO SCIENTIFIC VISUALIZER

‘‘Darkness fell in from every side, a sphere of singing black, pressure on the extended crystal nerves of the universe of data he had nearly become... And when he was nothing, compressed at the heart of all that dark, there came a point where the dark could be no *more*, and something tore. The Kuang program spurted from tarnished cloud, Case’s consciousness divided like beads of mercury, arcing above an endless beach the color of the dark silver clouds. His vision was spherical, as though a single retina lined the inner surface of a globe that contained all things, if all things could be counted. ’’

[Gibson, Neuromancer]

The Pseudo Scientific Visualizer is the object browser for the other half of your brain, a fish-eye lens for the macroscopic examination of data. It can display arbitrarily large, arbitrarily deep structures, in a fixed amount of space. It shows form, texture, density, depth, fan out, and complexity.

It draws a compound object as a circle, then recursively draws its elements, scaled smaller, in an evenly spaced ring, rotated around the circle. The deeper an object, the smaller it is. It will only draw to a certain depth, which you can change while the drawing is in progress.

It has simple graphical icons for different data types. An array is a circle, and a dictionary is a circle with a dot. The icon for a string is a line, whose length depends on the length of the string. A name is a triangle. A boolean is a peace sign or an international no sign. An event is an envelope. A process is a Porsche.

It randomly forks off several light weight processes, to draw different parts of the display, so there is lots of drawing going on in different places at once, and the overlapping is less regular.

After the drawing is complete, the circular compound objects become mouse sensitive, selectable targets. The targets are implemented as round transparent NeWS canvases. When you move the cursor over one, it highlights, and you can click on it to zoom in, pop up a description of it, open up another view of it, or select it, and then push it onto the stack of the PSIBER Space Deck.

Figure 8 shows a Pseudo Scientific visualization of the NeWS rootmenu instance dictionary, also shown in figure 3 and figure 7. Figure 9 shows two views of a map of the ARPAnet. Figure 10 shows two views of a map of Adventure.

## 8. REFERENCES

Abelson, Harold; Sussman, Gerald
Structure and Interpretation of Computer Programs; 1985; The MIT Press, Cambridge, Mass. and McGraw Hill, New York

Adobe Systems
PostScript Language Tutorial and Cookbook (The Blue Book); 1985; Addison-Wesley Publishing Company, Inc., Reading, Mass.

PostScript Language Reference Manual (The Red Book); 1985; Addison-Wesley Publishing Company, Inc., Reading, Mass.

Adobe Systems; Reid, Glenn C.
PostScript Language Program Design (The Green Book); 1988; Addison-Wesley Publishing Company, Inc., Reading, Mass.

Bier, Eric A.; Stone, Maureen
Snap-dragging; SIGGRAPH’86 Proceedings; Page 233-240; 1986; ACM, New York

Callahan, Jack; Hopkins, Don; Weiser, Mark; Shneiderman, Ben
A Comparative Analysis of Pie Menu Performance; Proc. CHI’88 conference, Washington D.C.; 1988; ACM, New York

Densmore, Owen
Object Oriented Programming in NeWS; November 1986; USENIX Monterey Computer Graphics Workshop; Usenix Association

Gibson, William
Neuromancer; 1984; ACE Science Fiction Books; The Berkley Publishing Group, New

York

Gosling, James; Rosenthal, David S.H.; Arden, Michelle
The NeWS Book; 1989; Springer-Verlag, New York

Goswell, Crispin
‘‘An Implementation of PostScript’’, in Workstations and Publication Systems; Rae A. Earnshaw, Editor; 1987; Springer-Verlag, New York

Hopkins, Don
‘‘Directional Selection is Easy as Pie Menus!’’, in ;login: The USENIX Association Newsletter; Volume 12, Number 5; September/October 1987; Page 31

A Pie Menu Cookbook: Techniques for the Design of Circular Menus; (Paper in preparation. Draft available from author.)

Hopkins, Don; Callahan, Jack; Weiser, Mark
Pies: Implementation, Evaluation, and Application of Circular Menus; (Paper in preparation. Draft available from authors.)

Reid, Glenn
The Distillery (program); available from ps-file-server@adobe.com

Shu, Nan C.
Visual Programming; 1988; Van Nostrand Reinhold; New York

Sun Microsystems
NeWS 1.1 Manual; 1987; Sun Microsystems; Mountain View, California

## 9. ACKNOWLEDGMENTS

This work could not have been done without the greatly appreciated support of the University of Maryland Human-Computer Interaction Lab, Grasshopper Group, Sun Microsystems, and NCR Corporation.

Many thanks to Owen Densmore, Hugh Daniel, Mike Gallaher, John Gilmore, James Gosling, Marja Koivunen, David LaVallee, Julia Menapace, Rehmi Post, Brian Raymor, Glenn Reid, David Rosenthal, Robin Schaufler, Ben Shneiderman, Josh Siegel, Stan Switzer, and Martha Zimet, people who gave me much valuable advice, encouragement, feedback, and lots of neat ideas.