

The Smalltalk Environment

Larry Tesler
Apple Computer Inc
10260 Bandley Dr
Cupertino CA 95014

As I write this article, I am wearing a T-shirt (photo 1) given to me by a friend. Emblazoned across the chest is the loud plea:

DON'T
MODE
ME IN

Surrounding the caption is a ring of barbed wire that symbolizes the trapped feeling I often experience when my computer is "in a mode."

In small print around the shirt are the names of some modes I have known and deplored since the early 1960s when I came out of the darkness of punched cards into the dawn of interactive terminals. My rogues' gallery of inhuman factors includes command modes like INSERT, REPLACE, DELETE, and SEARCH, as well as that inescapable prompt, "FILE NAME?" The color of the silk screen is, appropriately enough, very blue.

My friend gave me the shirt to make fun of a near-fanatical campaign I have waged for several years, a campaign to eliminate modes from the face of the earth—or at least from the face of my computer's display screen. It started in 1973 when I began work at the Xerox Palo Alto Research Center (PARC) on the design of interactive systems to be used by office workers for document preparation. My observations of secretaries learning to use the text editors of that era soon convinced me that my beloved computers were, in fact, unfriendly monsters, and that their sharpest fangs were the ever-present modes. The most common question asked by new users,

at least as often as "How do I do this?," was "How do I get out of this mode?" Other researchers have also condemned the prevalence of modes in interactive systems for novice users (reference 1).

Novices are not the only victims of modes. Experts often type commands used in one mode when they are in another, leading to undesired and distressing consequences. In many systems, typing the letter "D" can have meanings as diverse as "replace the selected character by D," "insert a D before the selected character," or "delete the selected character." How many times have you heard or said, "Oops, I was in the wrong mode"?

Preemption

Even when you remember what mode you are in, you can still fall into a trap. If you are running a data-plotting program, the only commands you can use are the ones provided in that program. You can't use any of the useful capabilities of your computer that the author of the program didn't consider, such as obtaining a list of the files on the disk. On the other hand, if you're using a program that lets you list files, you probably can't use the text editor to change their names. Also, if you are using a text editor, you probably can't plot a graph from the numbers that appear in the document.

If you stop any program and start another, data displayed by the first program is probably erased from the screen and irretrievably lost from view. In general, "running a program" in most systems puts you into a mode where the facilities of other programs are unavailable to you. Dan Swinehart calls this the *dilemma of preemption* (reference 2).

Many systems feature hierarchies of modes. A portion of a typical mode hierarchy is shown in figure 1. If you are in the editor and want to copy text from a file, you issue the *copy-from* command and it gives the prompt "from what file?" You then type a file name. What if you can't remember the spelling? No problem. Leave *from-what-file* mode, leave *copy-from* mode, save the edited text, exit from the editor to the executive, call up *file management* from the executive, issue the *list-files* command, look for the name you want (Hey, that went by too fast. Sorry, you can't scroll backwards in that mode.), terminate the *list* command, exit from *file management* to the executive, reenter the *editor*, issue the *copy-from* command, and when it prompts you with "from-what-file?", simply type the name (you haven't

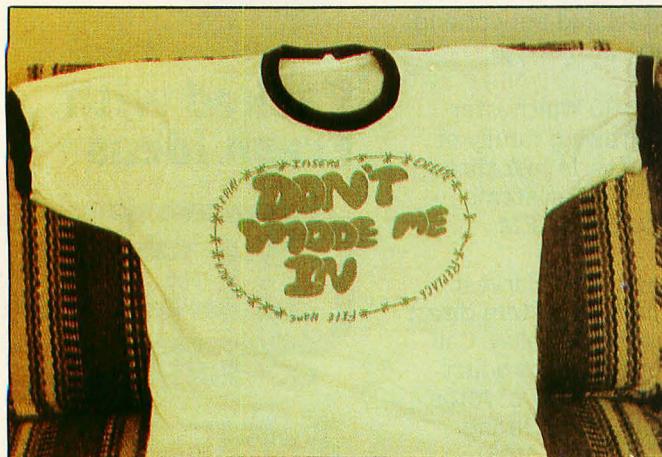


Photo 1: The "DON'T MODE ME IN" T-shirt.

forgotten it, have you?).

You don't have to be a user-sympathizer to join the campaign against modes. The most coldhearted programmer is a victim as well. Say you have programmed a new video game for your personal computer and have encountered a bug. An obscure error message appears on the screen mixed in with spacecraft and alien forms. To see the latest version of the program on the screen, you have to wipe out the very evidence you need to solve the problem. Why? Because the system forces you to choose between edit mode and execute mode. You can't have both.

Enter the Integrated Environment

Soon after I began battling the mode monster, I became associated with Alan Kay, who had just founded the Learning Research Group (LRG) at the Xerox PARC. Kay shared my disdain for modes and had devised a user-interface paradigm (reference 3) that eliminated one kind of mode, the kind causing the preemption dilemma. The paradigm he advocated was called "overlapping windows."

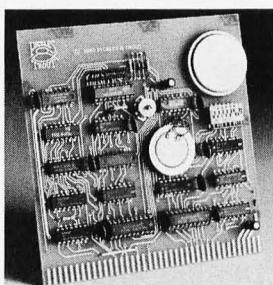
Most people who have used computer displays are familiar with windows. They are rectangular divisions of the screen, each displaying a different information set. In some windowing systems, you can have several tasks in progress, each represented in a different window, and can switch freely between tasks by switching between windows.

What is a

CLOCALPEEP?

Another name for the CCB-II, which is:

- a clock
hour, minute, second
- a calendar
day, day of week,
month, year
- an audio alarm



All on one board for your

TRS-80 Model II

It includes a pacemaker battery which will give over 8 years of continuous timekeeping.

From the folks who brought you the best CP/M® for the Model II.

\$175 plus shipping

Prepaid, COD, Mastercharge or Visa orders accepted. California residents add 6% sales tax.

TRS-80 is a trademark of Tandy Corp.

CP/M is a registered trademark of Digital Research Inc.



PICKLES & TROUT
P.O. BOX 1206, GOLETA, CA 93116, (805) 967-9563

Warning: Installation requires opening the Model II, which may void its warranty. We suggest that you wait until the warranty period has expired before installing the CCB-II.

The trouble with most windowing systems is that the windows compete with each other for screen space—if you make one window bigger, another window gets smaller. Kay's idea was to allow the windows to overlap. The screen is portrayed as the surface of a desk, and the windows as overlapping sheets of paper (photo 2). Partly covered sheets peek out from behind sheets that obscure them. With the aid of a pointing device that moves a cursor around the screen, you can move the cursor over a partly covered sheet and press a button on the pointing device to uncover that sheet.

The advantages of the overlapping-window paradigm are:

- the displays associated with several user tasks can be viewed simultaneously
- switching between tasks is done with the press of a button
- no information is lost switching between tasks
- screen space is used economically

Of course, windows are, in a sense, modes in sheep's clothing. They are more friendly than modes because you can't slip into a window unknowingly when you are not looking at the screen, and because you can get in and out of any window at any time you choose by the push of a button.

Kay saw his paradigm as the basis for what he called an "integrated environment." When you have an integrated environment, the distinction between operating system and application fades. Every capability of your personal computer is always available to you to apply to any information you want. With minimal effort, you can move among such diverse activities as debugging programs, editing prose, drawing pictures, playing music, and running simulations. Information generated by one activity can be fed to other activities, either by direct user interaction or under program control.

When Kay invented the Smalltalk language in 1972, he designed it with the ability to support an integrated en-

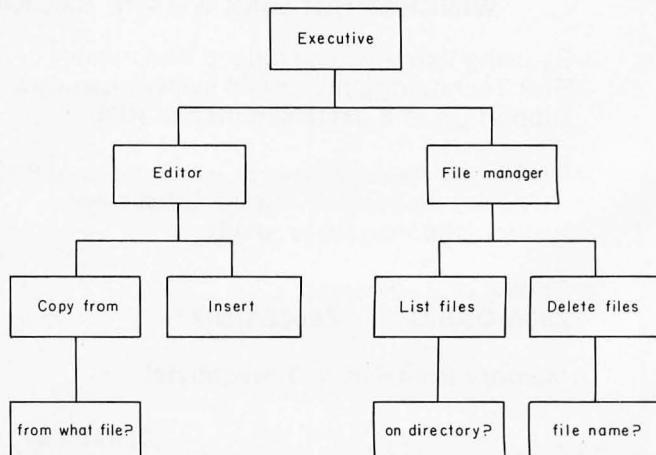


Figure 1: A portion of a typical mode hierarchy.

vironment. The implementations of Smalltalk produced by Dan Ingalls and the other members of the Learning Research Group have achieved ever-increasing integration. The file system, process-management system, graphics capability, and compiler are implemented almost entirely in Smalltalk. They are accessible from any program, as well as by direct user interaction.

In recent years, the idea of an integrated environment has spread outside the Learning Research Group and even to non-Smalltalk systems. The window-per-program paradigm is now commonplace, and many system designers have adopted the overlapping-sheet model of the screen.

In summary, the term *environment* is used to refer to everything in a computer that a person can directly access and utilize in a unified and coordinated manner. In an *integrated environment*, a person can interweave activities without losing accumulated information and without giving up capabilities.

Strengths of Smalltalk

Before delving further into the *nature* of the Smalltalk environment, we should first discuss its *purpose*.

Many general-purpose programming languages are more suitable for certain jobs than others. BASIC is easy to learn and is ideal for small dialogue-oriented programs. FORTRAN is well suited to numerical applications. COBOL is tailored to business data processing. Pascal is good for teaching structured programming.

LISP is wonderful for processing symbolic information. APL excels at manipulating vectors and matrices. C is great for systems programming. SIMULA shines at discrete simulations. FORTH lets people quickly develop efficient modular programs on very small computers.

All these languages have been used for numerous purposes in addition to those mentioned. You can write almost any program better in a language you know well than in one you know poorly. But if languages are compared from a viewpoint broader than that of a narrow expert, each language stands out above the others when used for the purpose for which it was designed.

Although Smalltalk has been used for many different applications, it excels at a certain style of software development on a certain type of machine. The machine that best matches Smalltalk's strengths is a personal computer with a high-resolution display, a keyboard, and a pointing device such as a *mouse* or graphics tablet (photo 3a). A cursor on the screen tracks mouse movements on the table so you can point to objects on the screen. The mouse (reference 4) has one or more buttons on its top side (photo 3b). One button is used as a *selection button*. If there are more buttons, they are normally used as *menu buttons*.

If the machine has a high-performance disk drive, you can use a virtual-memory version of Smalltalk and have as little as 80 K bytes of main memory, not counting display-refresh memory. Otherwise, you should have at least 256 K bytes of memory. This much memory is required because the whole integrated environment lives in one address space. It includes not only the usual run-time language support, but window-oriented graphics, the

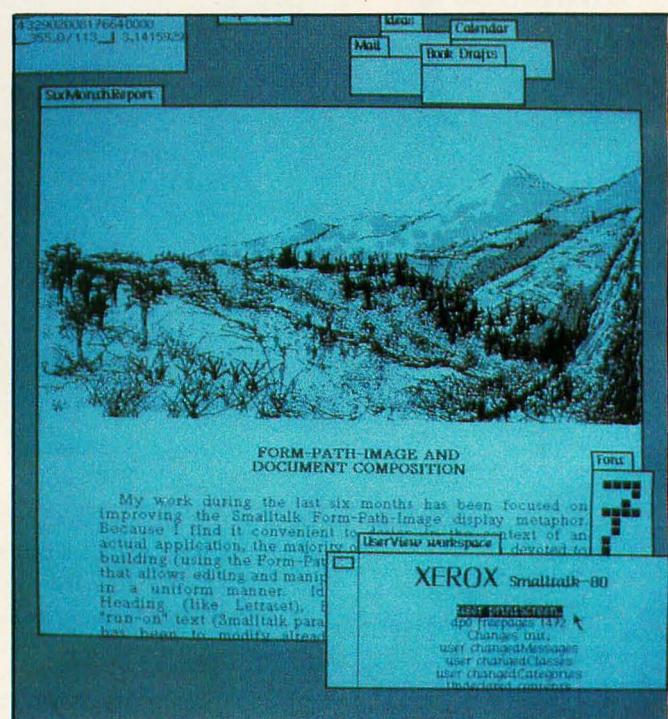


Photo 2: A typical Smalltalk display. The various "windows" look and behave like overlapping sheets of paper.

FREE

Computer Forms Catalog

**with 32 pages of
continuous
business forms for
small computer
systems**

Send today for our NEW full color 32 page catalog with programming guides, prices and order forms for continuous checks, invoices, statements, envelopes, stock paper and labels.

- Quality products at low prices
- Available in small quantities
- Fast Service
- Money Back Guarantee
- Convenient TOLL-FREE ordering

Fast Service by mail or... PHONE TOLL FREE

1 + 800-225-9550

Mass. residents 1 + 800-922-8560

8:30 a.m. to 5:00 p.m. Eastern Time Monday — Friday

Please rush a new computer forms catalog to:

CODE 20460

Name _____

Company _____

Street _____

City, State and Zip _____

Phone _____

Computer make & model _____



**Nebs
Computer Forms**

78 Hollis Street, Groton, Mass. 01471
A division of New England Business Service, Inc.

editor, the compiler, and other software-development aids. The programs you write tend to be small because they can build on existing facilities; no system facilities are hidden from the user. Users of LISP and FORTH will be familiar with this idea.

Smalltalk supports its preferred hardware by incorporating software packages that provide:

- output to the user through overlapping windows
- input from a keyboard, a pointing device, and menus
- uniform treatment of textual, graphical, symbolic, and numeric information

These interactive facilities are utilized heavily by the built-in programming aids and are available to all user-written applications.

The style of software development to which Smalltalk is oriented is *exploratory*. In exploratory development, it should be fast to create and test prototypes, and it should be easy to change them without costly repercussions. Smalltalk is helpful because:

- The language is more concise than most, so less time is spent at the keyboard.

(3a)



(3b)



Photo 3: A typical Smalltalk system (photo 3a) and a close-up of the "mouse" (photo 3b), a device that allows you to move an on-screen cursor and select certain options.

- The text editor is simple, modeless, and requires a minimum of keystrokes.
- The user can move among programming, compiling, testing, and debugging activities with the push of a button.
- Any desired information about the program or its execution is accessible in seconds with minimal effort.
- The compiler can translate and relink a single change into the environment in a few seconds, so the time usually wasted waiting for recompilation after a small program modification is avoided.
- Smalltalk programs grow gracefully. In most environments, a system gets more difficult to change as it grows. If you add 2 megabytes of virtual memory to the Smalltalk environment, you can fill the second megabyte with useful capabilities as fast as you can fill the first.
- The class structure of the language prevents objects from making too many assumptions about the internal behavior of other objects (see David Robson's article, "Object-Oriented Software Systems," on page 74 of this issue). The programmer can augment or change the methods used in one part of a program without having to reprogram other parts.

The Anatomy of a Window

Over the years, members of the Learning Research Group have embellished Kay's original window concept. Let us look at a Smalltalk window in more detail (figure 2).

The window is shown as a *framed rectangular area* with a *title tab* attached to its top edge. The program associated with the window must confine its output to the framed area.

Every window has a *window menu* (photo 4a). The window menu includes commands to reframe the window in a new size and location, to close the window, to print the contents of the window on a hard-copy device, and to retrieve windows hidden under it.

A window is tiled by one or more *panes*, each with its own *pane menu* (photo 4b). The pane menu includes commands appropriate to the contents of that pane. In

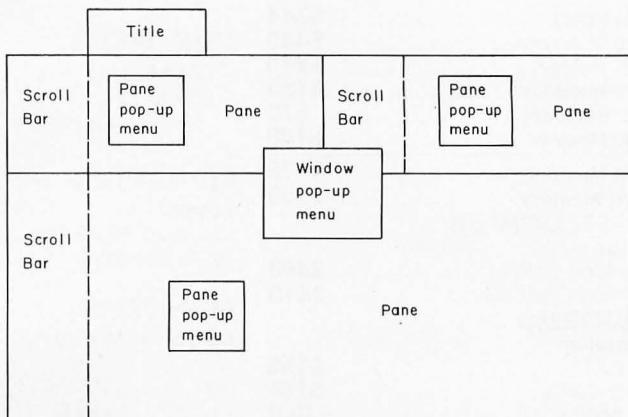


Figure 2: Anatomy of a window.

CAT-100 FULL COLOR GRAPHICS

Complete line of color imaging systems with high resolution real time video FRAME GRABBER for the S-100 bus.

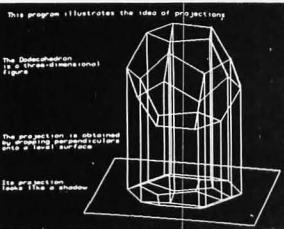
Capture and digitize a video frame in 1/60th of a second. Store up to 2 million bits of image data in on-board buffer. By software, select the best resolution for your application from 256 to 1280 pixels per TV line. Display your digitized image or your computer processed image with up to 256 gray levels or 65,536 simultaneous colors on standard B/W, NTSC or RGB color TV monitors.



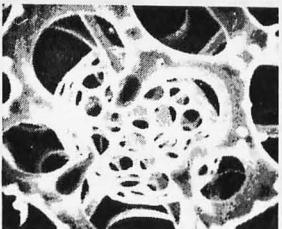
480x512 Contoured digitized image



240x256 Digitized image, 16 levels



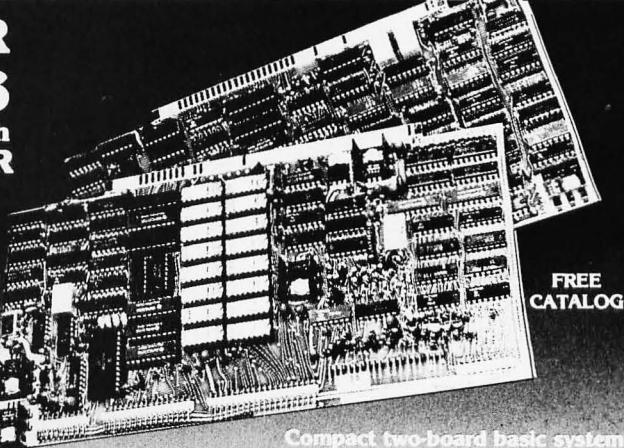
480x512 Computer-generated



240x256 Digitized image, 16 levels

This program illustrates the idea of projection. The three-dimensional figure is projected onto a two-dimensional surface. The projection is obtained by mapping the colors of the figure onto the surface. The projection casts a shadow.

ROBOTICS-AUTOMATED COUNTING AND MEASURING-QUALITY CONTROL-MONITORING-PATTERN RECOGNITION-IMAGE PROCESSING-MEDICAL AND SCIENTIFIC RESEARCH



FREE CATALOG

Compact two-board basic system

Features:

- Highest quality 480x512x8 digital video image
- Input capability from TV cameras or other sources
- Variety of synchronization choices
- 2 selectable video A/D conversion circuits
- Choice of 1, 2, 4, 8, 16 or 24 bits per pixel
- 32K-byte image memory on the basic system
- 64K, 128K, 192K and 256K-byte system capacity
- High resolution lightpen input
- Photographic trigger control input
- Software selectable system parameters
- Stand-alone intelligent graphics unit with universal interface

Accessories:

- RGB color monitor
- High performance lightpen
- Hard copy printer/plotter
- Software packages available



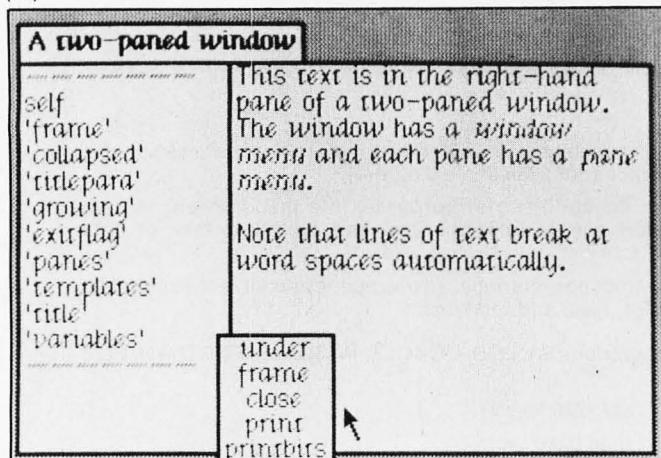
DIGITAL
GRAPHIC
SYSTEMS

935 Industrial Avenue
Palo Alto, CA 94303
(415) 856-2500

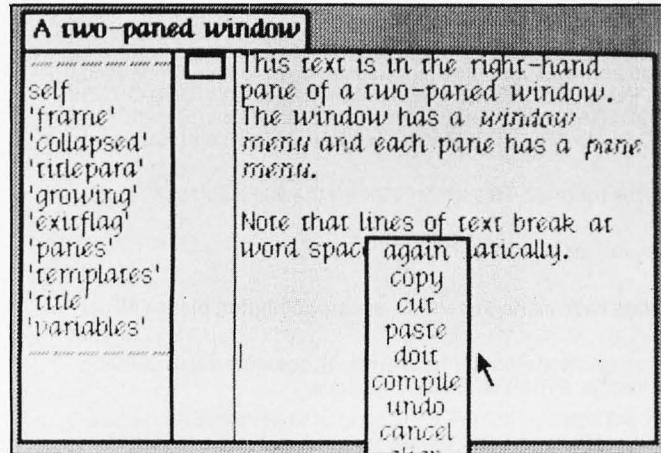
addition, a pane has a *scroll bar* on its left side used to scroll the contents of the pane when more information exists than fits in the frame at one time.

Although you can see many windows and panes at once, you can interact with only one pane at a time. That pane and its window are said to be awake or *active*. To awaken a different pane of the same window, move the cursor over the new pane (photo 4c). To awaken a different window, move the cursor over the new window and press the *selection button* on the pointing device (photo 4d). When a window wakes up, its title tab and all

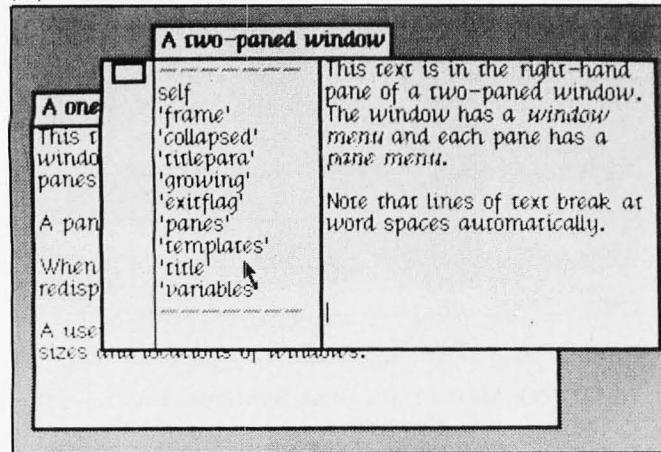
(4a)



(4b)



(4c)



its panes are displayed, and it is no longer covered up by other windows.

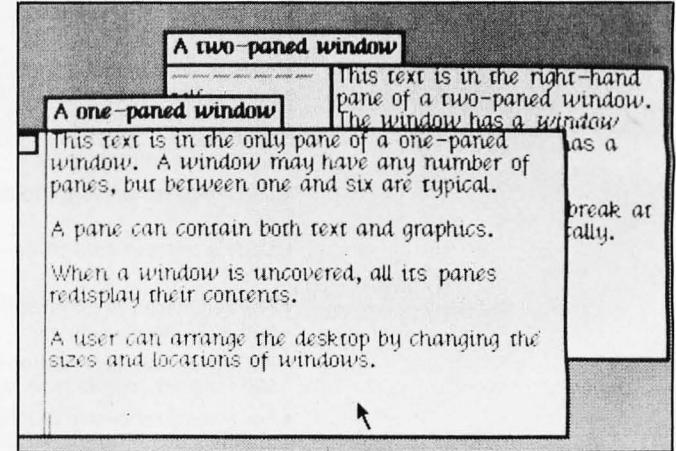
The scroll bar of the active pane is called the *active scroll bar*. Its menu and the menu of its window are called the *active menus*. In order to reduce screen clutter and maximize utilization of precious screen space, no inactive scroll bars or menus are displayed. On machines that use a pointing device with three buttons, some versions of Smalltalk even hide the active menus until one of two *menu buttons* is pressed, at which time the associated menu *pops up* and stays up until the button is released. If the button is released when the cursor is over a command in the menu, that command is executed (photo 4e).

Modeless Editing

The overlapping-window paradigm helps eliminate preemption. It can also reduce the need for certain prompts and their associated modes. For example, you never have to type the name of a procedure you want to examine. At worst, you point to its name in a list; at best, the desired procedure is already in a window on the screen, and you activate that window.

Unfortunately, overlapping windows do not eliminate command modes like "insert" and "replace" by themselves. Between 1973 and 1975, I worked at PARC with various collaborators, including Dan Swinehart and Timothy Mott, to banish command modes from interac-

(4d)



(4e)

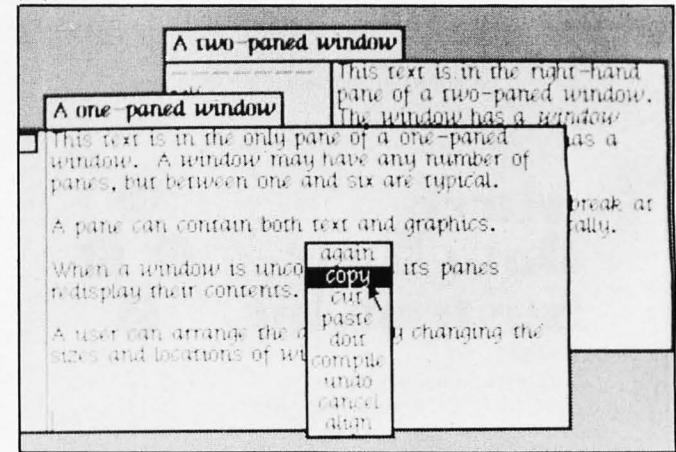


Photo 4: Windows and their behavior.

tive systems. Despite initial skepticism, nearly all users of our prototypes grew to appreciate the absence of modes. The following techniques were devised by us to eliminate modes from text editing. They are analogous to the techniques used to keep Polish-notation calculators relatively mode-free. Similar techniques can be applied to page layout, graphics creation, and other interactive tasks.

Selection precedes command:

- Every command is executed immediately when you issue it. You are not asked to confirm it. You can issue an *undo* command to reverse the effects of the last issued command. Although the main purpose of "undo" is to compensate for the lack of command confirmation, it can also be used to change your mind after issuing a command.
- For a command like "close the active window" that requires no additional parameters, you simply issue the command.
- For a command like "delete text" that requires one parameter, you first select the parameter using the pointing device and then issue the command. Until you issue the command, you can change your mind and make a different selection, or even choose a different command.
- For a command like "send electronic mail" that requires several parameters (recipient, subject, content), you first fill the parameters into a form using modeless text editing

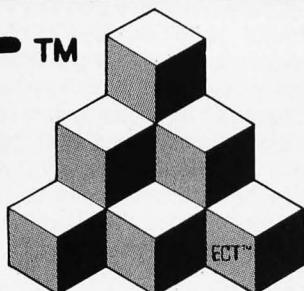
and then issue the command. You are not in a mode while filling out the form. If you want to copy something into the form from another place, you can. If you want to do something else instead, just do it; you may even return to the form later and finish filling it out.

Typing text always replaces the selected characters:

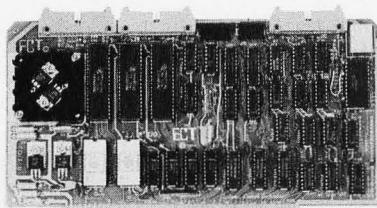
- Pressing a text key on the keyboard never issues a command. It always replaces the current selection by the typed character and automatically selects the gap following that character.
- To replace a passage of text, first select it (photo 5a) and then type the replacement. The first keystroke deletes the original text (photo 5b).
- To insert between characters, you first select the gap between those characters (photo 6a) and then type the insertion (photo 6b). Essentially, you are replacing nothing with something.
- The destructive backspace function always deletes the character preceding the selection, even if that character was there before the selection was made.
- The "undo" command (photos 6c and 6d) can be used to reverse the effects of all your typing and backspacing since you last made a selection with the pointing device.

Thus, the usual insert, append, and replace modes are folded into one mode—replace mode—and one mode is no mode at all.

ECT™



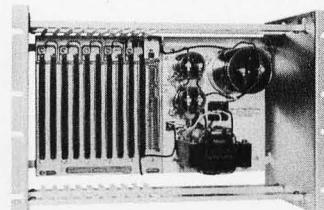
R²I/O
S-100 ROM,
RAM & I/O
BOARD



ECT's R²I/O is an S-100 Bus I/O Board with 3 Serial I/O Ports (UART's), 1 Parallel I/O Port, 4 Status Ports, 2K of ROM with the 8080 Apple Monitor Program and 2K of Static RAM.

\$295.00

Building Blocks for Microcomputer Systems, Dedicated Controllers and Test Equipment.



RM-10
S-100
RACK MOUNT
CARD CAGE

ECT's RM-10 is a rack mount 10 slot Card Cage with Power Supply, consisting of an ECT-100 rack mount Card Cage (19"W x 12.25"H x 8"D), the MB-10 Mother Board (with ground plane and termination) all 10 connectors and guides and the PS-15A Power Supply (15A @ 8V, 1.5A @ ± 16V).

\$295.00

Specializing in Quality Microcomputer Hardware
Industrial • Educational • Small Business • Personal

Card Cages, Power Supplies, Mainframes, CPU's, Memory, I/O, OEM Variations

ECT™

ELECTRONIC CONTROL TECHNOLOGY (201) 686-8080

763 Ramsey Ave., Hillside, NJ 07205

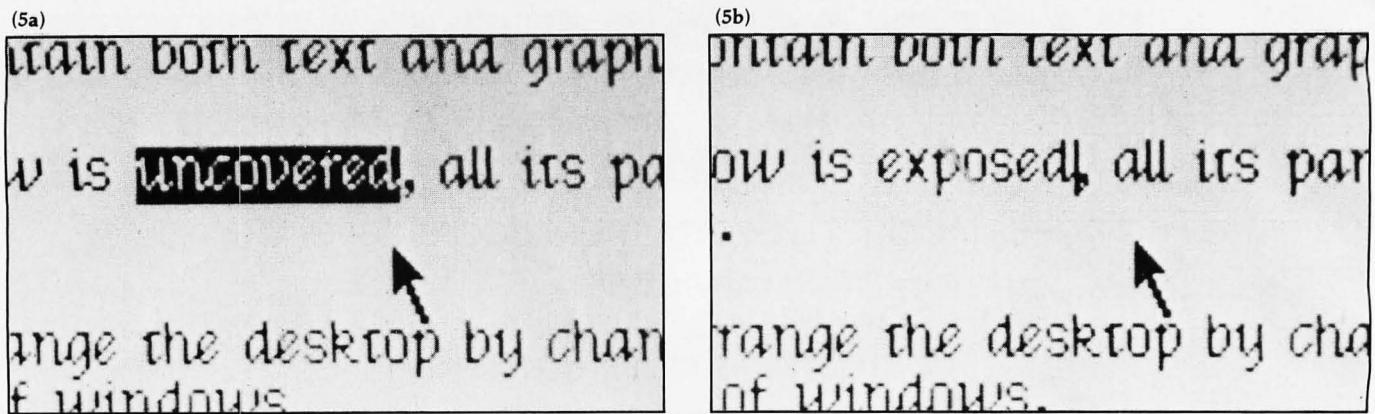


Photo 5: Replacing text in Smalltalk.

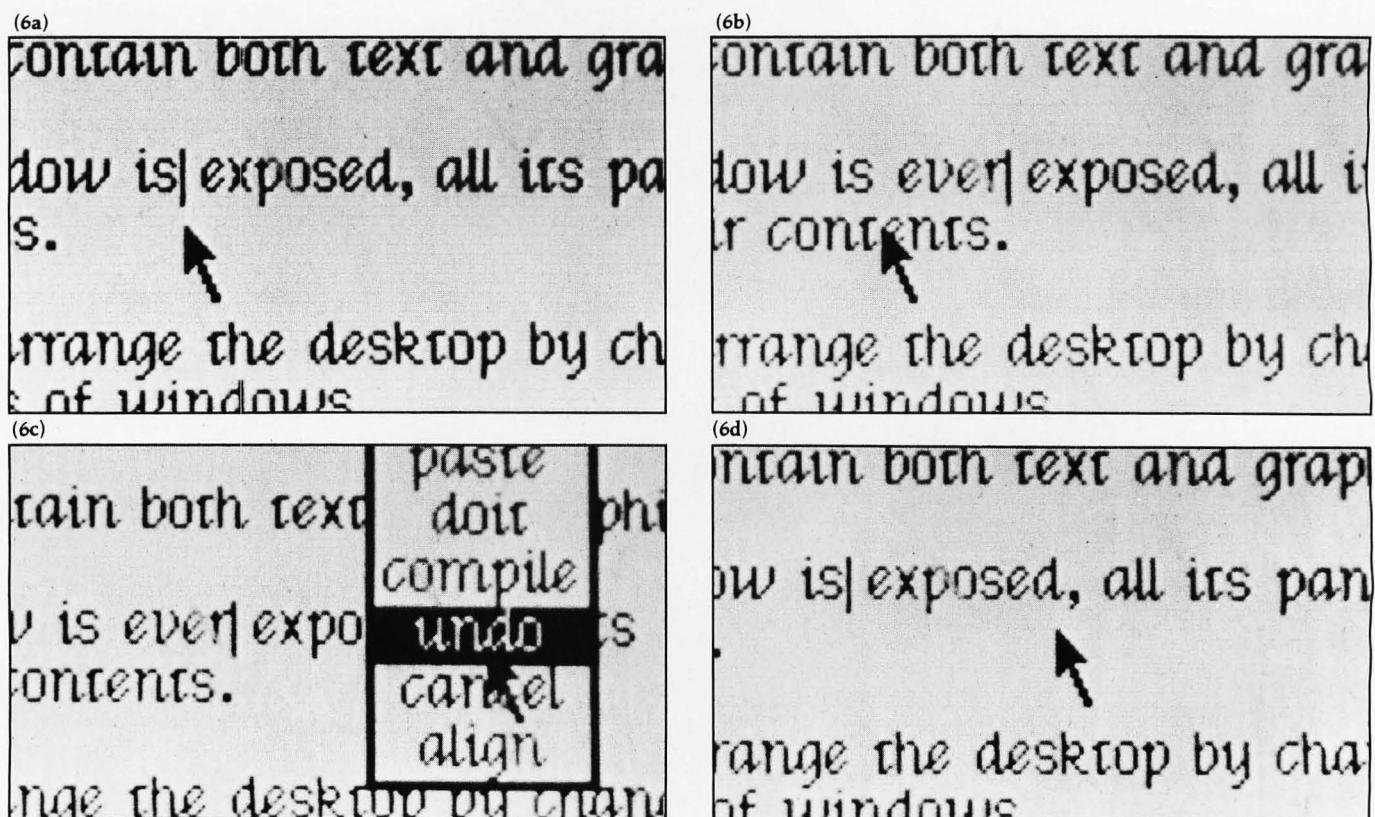


Photo 6: Inserting text in Smalltalk.

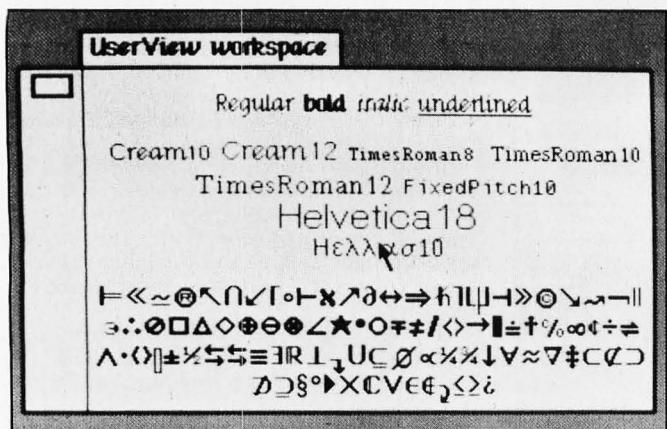


Photo 7: Multiple typefaces can be used in any window.

The "shift lock" key and analogous commands like "bold shift" and "underline shift" cause modes for the interpretation of subsequently typed characters. However, shifts are familiar to people and are relatively harmless. The worst they do is change a "d" to a "D," "d," or "d"—never to a Delete command.

The bit-map display can show boldface characters, as well as italics, underlining, and a variety of styles and sizes of printer's type (photo 7). Thus, as you enter text in bold shift, the screen shows what the text will look like when it is printed. A command like bold shift can also be applied to existing text to change it to boldface.

In 1976, Dan Ingalls devised a user interface for Smalltalk that incorporated most of the mode-avoidance techniques discussed earlier. Consequently, it is rare in

the present Smalltalk environment to encounter a mode.

Making a Selection

In the Smalltalk-76 user interface, text is selected using the pointing device and a single button. First, the cursor is moved to one end of the passage to be selected (photo 8a). The selection button is pressed and held down while the cursor is moved to the other end of the passage. This operation is called "draw-through," though it is not necessary to traverse intermediate characters en route to the destination. When the cursor reaches the other end of the passage, the button is released. The selected passage is then shown in inverse video (photo 8d).

The feedback given to the user during selection is as follows. When the button is depressed, a vertical bar appears in the nearest intercharacter gap (photo 8b). (At the

(8a)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

(8b)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

(8c)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

left end of a line of text, the bar appears to the left of the first character. At the right end of a line, the bar appears to the left of the final space character.)

If the button is released without moving the cursor, the bar remains, indicating that a zero-width selection has been made. This method—clicking once between characters—is the one to use before you insert new text.

If the button is held down while the cursor is moved, the system supplies continuous feedback by highlighting in inverse video all characters between the initial bar and the gap nearest to the cursor (photo 8c). When the button is released, the selected characters remain highlighted (photo 8d). This method—drawing through a passage—is the one to use before you copy, move, delete, or replace text, or before you change it to boldface or otherwise alter its appearance.

Clicking the button twice with the cursor in the same spot within a word selects that whole word and highlights it (photo 8e). This special mechanism is provided because it is very common to select a word. Informal experiments lead us to believe that double clicking is much easier than drawing through a word for beginners and experts alike. It is also faster. It takes the average user about 2.6 seconds to select a word anywhere on the screen using draw-through, but it takes only 1.5 seconds using the double click (reference 5).

There is only one selection in the active pane. It is called the *active selection*.

(8d)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

(8e)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

When a window is exposed, all its panes redisplay their contents.

Photo 8: Selecting text using the mouse and the cursor.

SOFTWARE for the ATARI 800* and ATARI 400* from QUALITY SOFTWARE



add to the enjoyment of this program. At least 24K of RAM is required.
On Cassette — \$19.95 On Diskette — \$22.95

NAME THAT SONG

By Jerry White

Here is great entertainment for everyone! Two players listen while the Atari starts playing a tune. As soon as a player thinks he knows the name of the song, he presses his assigned key or joystick button. There are two ways to play. The first way requires you to type in the name of the song. Optionally, you can play multiple choice, where the computer asks you to select the title from four possibilities. The standard version requires 24K of RAM (32K on diskette) and has over 150 songs on it. You also get a 16K version that has more than 85 songs. The instructions explain how you can add songs to the program, if you wish. Written in BASIC.

On Cassette — \$14.95 On Diskette — \$17.95



QS FORTH

By James Albanese

Want to go beyond BASIC? The remarkably efficient FORTH programming language may be just for you. We have taken the popular fig-FORTH model from the FORTH Interest Group and expanded it for use with the Atari Personal Computer. Best of all we have written substantial documentation, packaged in a three ring binder, that includes a tutorial introduction to FORTH and numerous examples. QS FORTH is a disk based system that requires at least 24K of RAM and at least one disk drive. Five modules that may be loaded separately from disk are the fig-FORTH kernel, extensions to standard fig-FORTH, an on-screen editor, an I/O module that accesses Atari's operating system, and a FORTH assembler.

Diskette and Manual — \$79.95 Manual Only — \$39.95

FOR OUR COMPLETE LINE OF ATARI SOFTWARE
PLEASE WRITE FOR OUR CATALOG



QUALITY SOFTWARE
6660 Reseda Blvd., Suite 105, Reseda, CA 91335
(213) 344-6599

ASK FOR QUALITY SOFTWARE products at your favorite computer store. If necessary you may order directly from us. MasterCard and Visa cardholders may place orders by calling us at (213) 344-6599. Or mail your check or bankcard number to the address above. California residents add 6% sales tax. *Shipping Charges:* Within North America orders must include \$1.50 for shipping and handling. Outside North America the charge for airmail shipping and handling is \$5.00. Pay in U.S. currency.

*Indicates trademarks of Atari.

Issuing a Command

When you issue a command in Smalltalk, you are sending a message to an object. There are two ways to send a message from Ingalls's user interface. You can send certain commonly sent messages to the active pane or window by choosing them from menus; you can send any message to any object by direct execution of a Smalltalk statement.

(9a)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

When a window is exposed, all its panes redisplay their contents.

again	copy	th text and
copy	cut	graphics.
cut	doit	cancel
doit	compile	align
compile	undo	
undo	cancel	
cancel	align	

(9b)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes, but between one and six are typical.

A pane can contain both text and graphics.

When a window is exposed, all its panes redisplay their contents.

again	copy	th text and
copy	cut	graphics.
cut	doit	cancel
doit	compile	align
compile	undo	
undo	cancel	
cancel	align	

(9c)

A one-pane window

This text is in the only pane of a one-pane window. A window may have any number of panes.

A pane can contain both text and graphics.

When a window is exposed, all its panes redisplay their contents.

A user can arrange the desktop by

Photo 9: "Cutting" text in Smalltalk.

6809 SOFTWARE POWER TOOLS

BY MICROWARE®

OS-9™ MULTIPROGRAMMING OPERATING SYSTEM

A true multitasking, real time operating system for timesharing, software development, database, process control, and other general applications. This versatile OS runs on almost any 6809-based computer.

- UNIX™ -like file system with hierarchical directories, byte-addressable random-access files, and full file security. Versatile, easy-to-use input/output system is hardware independent and expandable.
- Powerful "shell" command interpreter features: I/O redirection, multiple job stream processing, and more. Includes a complete set of utility commands.
- OS-9 Level Two uses hardware memory management and can address over one megabyte of memory. Also includes pipes and filters for inter-process data transfers.
- OS-9 Level One runs on systems without memory management hardware having up to 56K memory.

□ OS-9 Level Two \$495* □ Level One \$195

BASIC09™ PROGRAMMING LANGUAGE SYSTEM

Extended BASIC language compiler/interpreter with integrated text editor and debug package. Runs standard BASIC programs or minimally-modified PASCAL programs.

- Permits multiple named program modules having local variables and identifiers. Modules are reentrant, position independent and ROMable.
- Additional control statements for structured programming: IF ... THEN ... ELSE, FOR ... NEXT, REPEAT ... UNTIL, WHILE ... DO, LOOP ... ENDLOOP, EXITIF ... ENDEXIT.
- Allows user-defined data types and complex data structures. Five built-in data types: byte, integer, 9 digit floating-point, string and boolean.

■ Runs under OS-9™ Level One or Level Two. □ \$195*

OTHER OS-9™ FAMILY SOFTWARE

- Stylograph™ Screen-Oriented Word Processor
- Interactive Assembler ■ Macro Text Editor
- Interactive Debugger

BASIC09 and OS-9 are trademarks of Microware® and Motorola. UNIX is a trademark of Bell Laboratories.* Most software is available on ROM or diskette in versions for many popular 6809 computers. Contact Microware® for specific availability.

For compatible hardware see GIMIX ad page 128.

MICROWARE®

Microware Systems Corp., Dept. B2
5835 Grand Avenue, Des Moines, Iowa 50304
(515) 279-8844 • TWX 910-520-2535

Smalltalk-76 provides pop-up menus for the most commonly used commands, like "cut," which deletes the selected text. To issue the "cut" command, you pop up the active-pane menu with one of the menu buttons on the mouse (photo 9a), keep that button down while moving the cursor to the command name (photo 9b), and then release the button (photo 9c). A command in the pane menu can have only one parameter, the active selection. A command in the window menu can have no parameters.

To issue a command that is not available in a menu, you select any place you can insert text, and type the whole command as a statement in the Smalltalk language (photo 10a). Then you select that statement and issue the

(10a)

A one-pane window

A user can arrange the desktop by changing the sizes and locations of windows.

4 max: 5



(10b)

A one-pane window

A user can arrange the desktop by changing the sizes and locations of windows.

again
copy
cut
paste
doit
compile
undo
cancel
align

4 max: 5

(10c)

A one-pane window

A user can arrange the desktop by changing the sizes and locations of windows.

4 max: 5



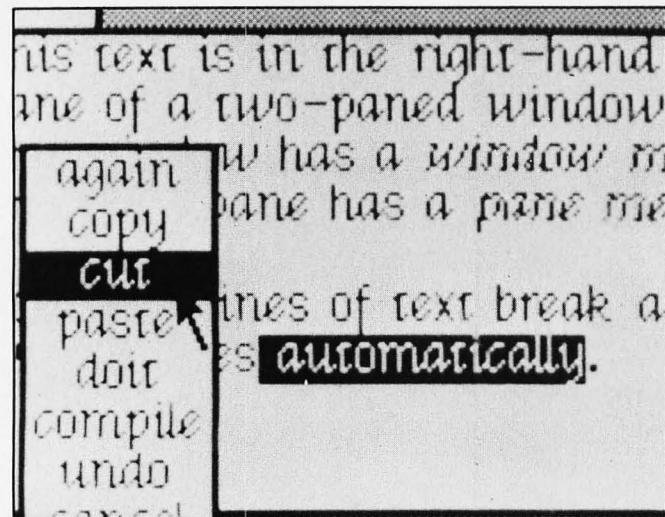
Photo 10: Executing text using the "doit" message.

single-parameter command "do it" (photo 10b) to obtain the result (photo 10c). The "do it" command provides immediate execution of any Smalltalk statement or group of statements. This method of command issuance uses the previous method: you are sending the message *doit* to the pane, with the Smalltalk statement as its parameter.

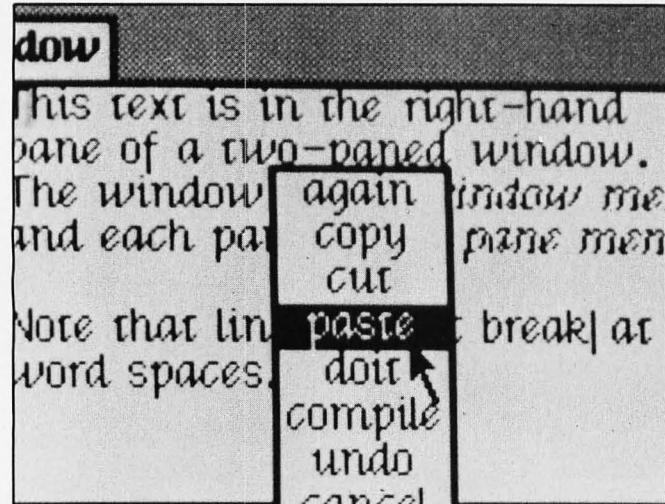
It is standard practice to keep a "work-space" window around the screen in which to type your nonmenu commands. When you want to reissue a nonmenu command issued earlier, simply select the command in the work-space window and "do it." You may, of course, edit some of the parameters of the old command before you select it and "do it." In a sense, you are filling out a form when you edit parameters of an immediate statement.

Unfortunately, the common commands "move text from here to there" and "copy text from here to there" cannot be issued by a single menu command because they require two parameters, the source selection and the destination selection. Sometimes, they even involve messages to more than one pane, the source pane and the destination pane. In a modeless system, a move or copy command is done in two steps:

(11a)



(11c)

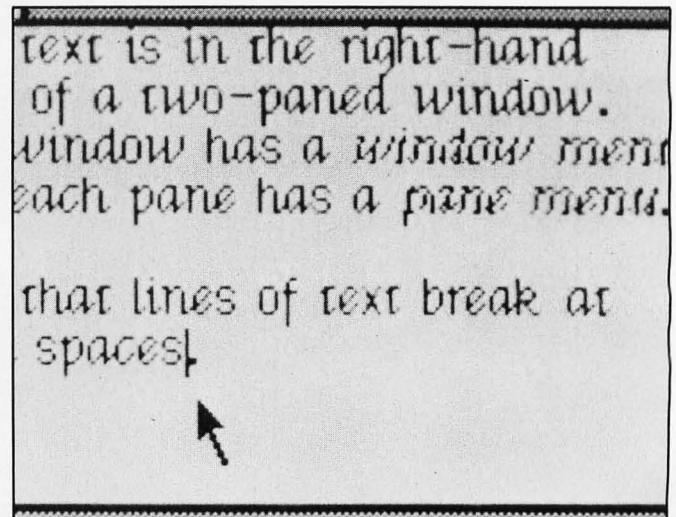


- A move is done by *cut and paste*. First, you select the source text and issue the "cut" command (photo 11a). The "cut" command deletes the selected text (photo 11b), but leaves it in a special place where it can be retrieved by "paste." Then you select the destination and issue the "paste" command (photo 11c) to complete the move (photo 11d).

- A copy is done by *copy and paste*, which is completely analogous to cut and paste, but does not delete the original text.

Remember the "copy-from-file" example (the one where you had to go in and out of many layers of modes)? In the Smalltalk-76 user interface, you can accomplish this with six pushed buttons, no mode exits, and no typing: (1) activate the source window that displays the file you are copying from; (2) select the desired text; (3) issue the "copy" command in the menu; (4) activate the destination window; (5) select the destination point, and (6) issue the "paste" command in the menu. The job requires little more effort than copying within the same document. If the window is not already

(11b)



(11d)

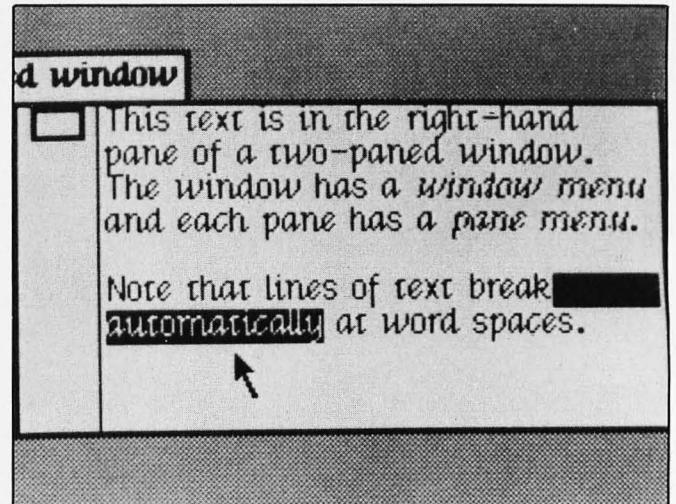


Photo 11: Moving text in Smalltalk.

on the screen and you can't remember the file name, you can go to another window and scroll through a list of files without having to exit any modes, invoke any programs, save any edits, lose sight of the destination file, or lose any time.

The Smalltalk-76 text-editing facilities not only relieve you of the burden of modes, they also require very few keystrokes and are easy to learn.

Software-Development Aids

One of my summer projects in 1977 was to increase the speed and friendliness of the Smalltalk software-development environment by adding *inspect windows*, *browse windows*, and *notify windows* to the user interface. These and other enhancements made by the Learning Research Group are described below. In recent months, the team has further enhanced the Smalltalk-80 environment. Although it conforms to the same principles as before, its details are different from what is described in this article.

Inspecting Data Structures

Suppose someone has given you a Smalltalk program to implement a "regular polygon" class (table 1) and you want to learn more about it. It would be helpful to see an actual instance of a regular polygon.

If the variable triangle refers to a regular polygon, you type the following statement into your work-space window:

```
triangle inspect
```

and then issue the "do it" command in the pane menu (photo 12a). In a few seconds, a two-paned "inspect window" appears on the screen. Its title tab tells you the class of the inspected object, in this case, RegularPolygon. The window is divided into two panes. The left or *variable pane* lists the parts of a regular polygon, sides, center, radius, and plotter. The right or *value pane* is blank.

You point to the word sides in the variable pane and click the selection button on the mouse. The word sides is highlighted, and in the value pane, the value of the variable sides appears (photo 12b), in this case, 3. You point to the word center and click. In the value pane appears the value of center (photo 12c), in this case, the point 526@302. The value pane is *dependent* on the variable pane because its contents are determined by what you select in the variable pane. The arrow in figure 3 symbolizes this dependency.

Let's inspect the value of center. In the variable pane, where center is selected, pop up the pane menu and issue

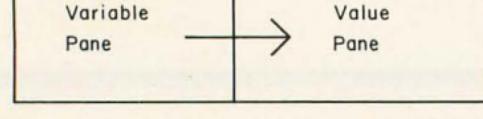


Figure 3: Principal dependencies among panes of an inspect window.

The following template contains a description of a regular polygon with the following attributes:

sides	Number of sides (3 for a triangle, 5 for a pentagon, etc.).
center	If the regular polygon were inscribed in a circle, this would be its center point.
radius	If the regular polygon were inscribed in a circle, this would be its radius.
plotter	A pen that can draw an image of the polygon on the screen or on paper.

The following expressions provide an example of creating and using an instance of RegularPolygon.

triangle ← RegularPolygon sides: 3 radius: 50.

triangle translateBy: -90 @ 60.
triangle plot: black.

triangle translateBy: 165 @ 20.
triangle scale: 0.6.
triangle plot: gray. "where gray denotes an ink color"

class name	RegularPolygon
superclass	Object
instance variable names	sides center radius plotter
class messages and methods	

initialization

sides: s radius: r | |

"Create an instance of RegularPolygon whose center is located at the center of the currently active window on the display screen. Screen is a global variable that refers to the hardware display screen."
t self new sides: s radius: r center: (Screen activeWindow frame center)

instance messages and methods

initialization

sides: s radius: r center: c | |

"Initialize all attributes. Class Pen is provided in the system as one way of side effecting the display screen."
sides ← s.
center ← c.
radius ← r.
plotter ← Pen width: 2

analysis

center | |

tcenter

sides | |

1sides

"Answer the center coordinate of the polygon."

"Answer the polygon's number of sides."

display

plot: Ink | |

plotter penup.

plotter goto: self center.

plotter up.

plotter go: radius.

plotter turn: 180 - (self cornerAngle/2).

plotter color: ink.

plotter pendn.

1 to: sides do:

[i | plotter go: self sideLength.

plotter turn: 180-self cornerAngle]

"Draw an image of the polygon using the specified ink color."

"lift the pen to disable drawing"

"position the pen at the center"

"face the top of the screen"

"position at a corner"

"turn to face along a side"

"select the ink color"

"lower the pen to enable drawing"

"for each side of the polygon:"

"plot that side"

"turn to face along the next side"

transformation

scale: factor | |

"Scale the polygon radius by the specified factor."

radius ← radius * factor

translateBy: deltaXY | |

"Change the polygon's location by the specified amount (a Point)."

center ← center + deltaXY

private instance methods

cornerAngle | | "Answer the interior angle of any vertex, in degrees."

180 - (360 / sides)

sideLength | | "Answer the length of any one of the equal sides."

12 * radius * (self cornerAngle / 2) degreesToRadians cos

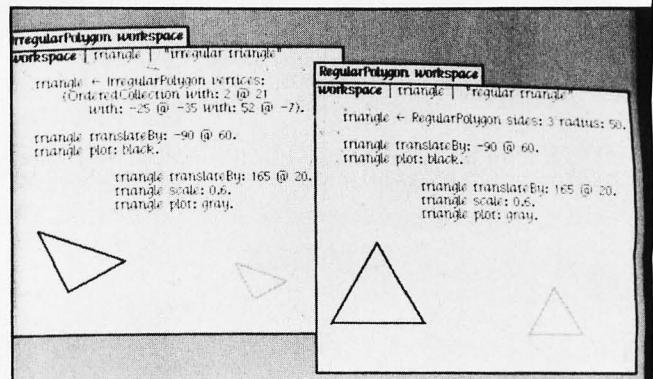
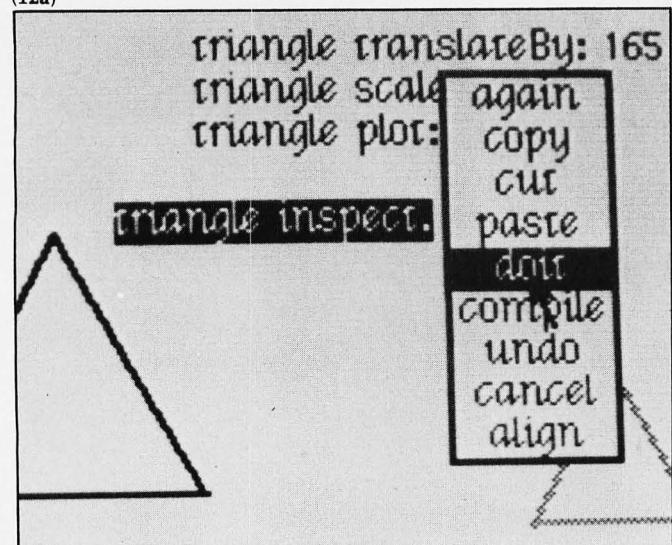
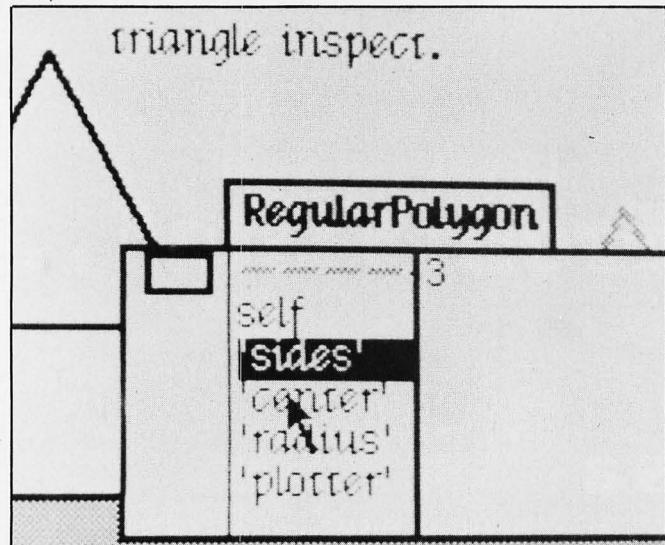


Table 1: Description and class template for class RegularPolygon.

the "inspect" command (photo 12d). On the screen appears another inspect window showing that center is an instance of class Point (photo 12e). You can now examine that point's variables, x and y, reactivate the original inspect window, close either or both windows, or work in any other window. You are not in a mode.



(12b)



(12c)

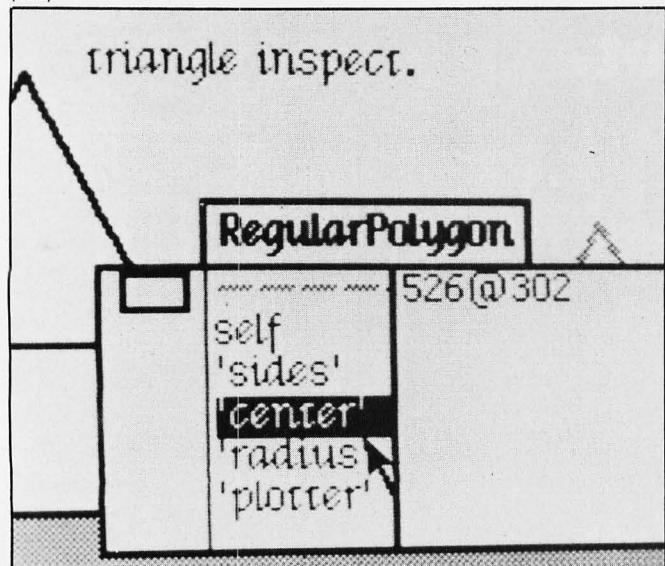


Photo 12: Inspecting data structures in Smalltalk.

amine that point's variables, x and y, reactivate the original inspect window, close either or both windows, or work in any other window. You are not in a mode.

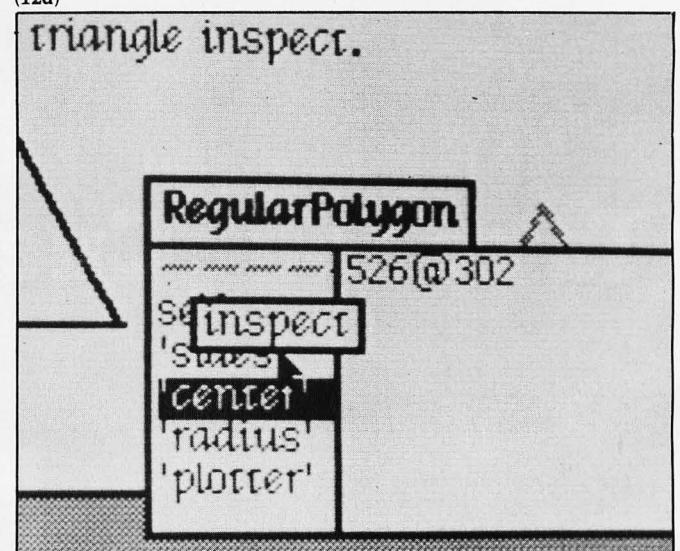
Browsing Through Existing Definitions

Now that you have inspected a sample regular polygon, you might want to find out what methods have been defined in its class. One way to do this is to activate a window called a "browse window" or "browser." Most Smalltalk programmers leave a browser or two on the screen at all times with the work-space window.

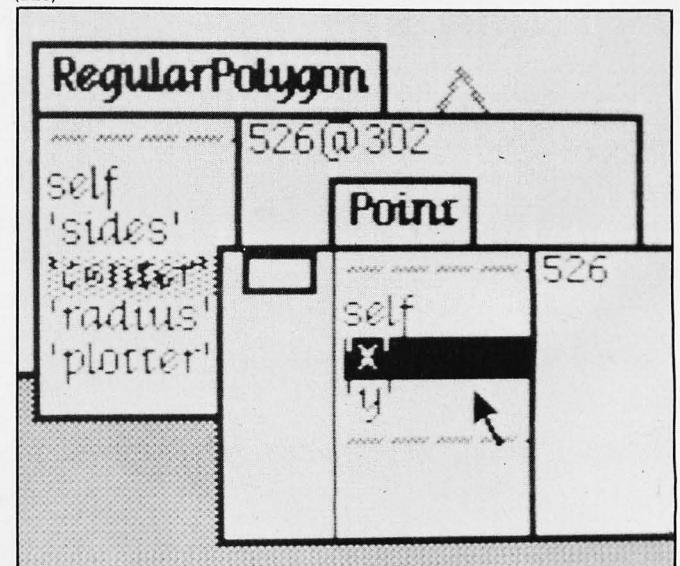
The title tab of the browser (photo 13a) says "Classes" because the standard browser lets you examine and change the definitions of all Smalltalk classes—classes supplied by the system, as well as classes supplied by yourself. It is easy to create a more restricted browser that protects the system from ill-conceived modification. But on a personal computer, you are just going to hurt yourself.

The browser has five panes. The principal dependence-

(12d)



(12e)



ADD EXCITING MUSIC TO YOUR APPLE®!



A L F's 9-voice Music Card MC1 is only \$195.

Here's what our customers have said:

(Excerpts from unsolicited letters. Copies of original letters available on request.)

About the A L F system:

It's a rare enough occurrence when hardware/software lives up to one's expectations. For something to exceed one's wildest hopes — as the A L F synthesizer certainly does — is a real treat. My congratulations to all concerned.

—Dhahran, Saudi Arabia

I myself have told several people that next to a disk, I consider the [A L F] synthesizer to be the most important peripheral they could purchase for their system. Very excellent job! Keep up the good work.

—Oak Ridge, Tennessee

I recently purchased 2 of your Apple music boards. Out of the peripherals I have for my Apple, I enjoy them the most. It has to be the most enjoyable thing that has ever been invented. I hope you continue to develop products as clever and enjoyable as this one. The Entry program has to be one of the most sophisticated programs I have ever seen. It proves that a hardware manufacturer DOES have the ability to also produce quality software. It is almost worth the price of the boards just for the Entry program.

—Burbank, California

About ease of use:

I have had my Music Card MC1 for a little more than a week now and I have almost completed entering "The Maple Leaf Rag". I found it to be a lot simpler than I thought and so I am very, very pleased. My family isn't because I sit up to all ends of the night playing with the blasted thing!

—Cypress, Texas

A L F has opened up my head and ears and enabled me to do things musically which I would like to be able to do on [conventional] instruments. As much as I love the instruments I try to play, I just don't have the talent and technique to play what is in my head. By golly, the A L F board doesn't know about my limitations, though. I can play hell out of that thing, playing notes and tempos which previously have existed only in my head. Many thanks from a frustrated musician and satisfied A L F "player".

—Demopolis, Alabama

About documentation:

I don't know much about hardware, but I have been a programmer for 15 years and I have never seen a better piece of software documentation than your user manual. It is a joy to study!

—Lancaster, California

About the competition:

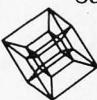
Recently, I purchased an [A L F] 9-voice board and a couple of music albums . . . all I can say is that I wish I had listened and played with it before I purchased the Mtn. Hardware board. It sounds about the same and is vastly superior in software, ease of use, and price. The Entry program is a joy to use and it's easier than Mtn. Hardware's, but then, I guess you guys know that already. (Oh yes, you wouldn't happen to know of anyone that wants to buy a Mtn. Hardware system? \$450 or best offer?)

—Kirkland, Washington

I would like to tell you that after having used the system ONLY ONE DAY, that I am absolutely delighted with it. In addition, I purchased the three boards although I ALREADY own Mountain Hardware's music system. Now that I have seen and own your system, I am putting my "old" one up for sale. I think that your software makes it far easier to enter music, and that the software routines allow for far greater flexibility. Again, I extend my compliments to you. As I said, I have owned another music system, and consider myself therefore, qualified to make a judgement between the use of the two. Yours is the clear choice!

—Levittown, New York

See your local Apple® dealer
or write:



A L F Products Inc.
1448 Estes Denver, CO 80215

Apple is a trademark of Apple Computer Inc.

cies between panes are symbolized by arrows in figure 4. The top row has four panes called the *class-category pane*, *class pane*, *method-category pane*, and *method pane*. The large lower pane is called the *editing pane*. (After you have used the system for a few minutes, the significance of each pane becomes apparent, and it is not necessary to memorize their technical names.)

In photo 13a, the browser shows a method definition in the editing pane. You can tell that the method is class RegularPolygon's version of scale: because RegularPolygon is highlighted in the class pane and scale: is highlighted in the method pane.

The method-category pane lists several groups of methods within class RegularPolygon: **initialization**, **analysis**, **display**, **transformation**, **testing**, and **private** methods. You can tell that scale: is a **transformation** message in class RegularPolygon because that category is highlighted.

The class-category pane lists several groups of classes, including **numbers**, **files**, and **graphical objects**. You can tell that class RegularPolygon is in the **graphical objects** group because that category is highlighted.

Suppose you want to look at a different method, translateBy:. Click its name in the method pane and its definition is immediately displayed in that pane's dependent, the editing pane (photo 13b). If the method you want to see is in the method category **analysis**, first click that category name. Immediately after you do that, its dependent, the method pane, lists the methods in that category. Now you can click the name of the desired method (photo 13c).

If you want to know things about the class as a whole, like its superclass and field names, click "Class Definition" in the method-category pane and the definition appears in the editing pane (photo 13d).

Suppose you want to look at a different class, say IrregularPolygon. Click its name in the class pane and its method categories are immediately displayed in the next pane (photo 13e). If the class you want to see is in the class category **windows**, first click that category name. Immediately after you do that, the class pane lists the classes in that category. Now you can click the name of

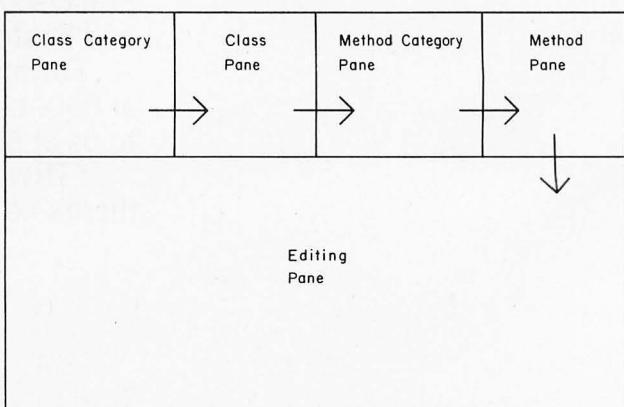


Figure 4: Principal dependencies among panes of a browse window.

the desired class (photo 13f).

Categorization is used at both the class and method level to help the programmer organize his or her program and to provide fewer choices in each pane. If a list is longer than what can fit in a pane, it can be scrolled by pressing a mouse button with the cursor in the scroll bar.

If you just want to browse around reading class and method definitions, you can do so by lazily clicking the selection button with the cursor over each name, never touching the keyboard. That is why the window is called a browser. Browsers are further discussed in references 6 and 7.

Astute readers may have noticed that the class template (see "The Smalltalk-80 System" by the Learning Research Group on page 36 of this issue) presents the methods of a class apart from the methods of its in-

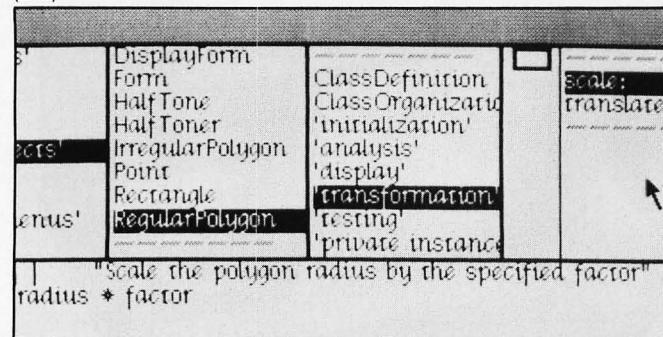
stances, while the browser does not. This discrepancy stems from differences between the Smalltalk-80 and Smalltalk-76 languages.

Revising Definitions

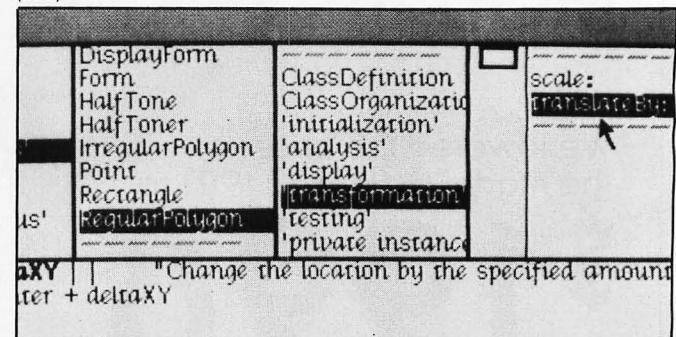
If you are looking at a method definition or class definition in the editing pane, you can revise it using the standard text-editing facilities (select, type, cut, paste, copy).

If you like, you can copy information into the definition from other windows—including other browse windows—because you are not in any mode while browsing. You can even interrupt your editing to run another program, list your disk files, draw a picture, or do whatever you like. You can later reactivate the browser and continue editing.

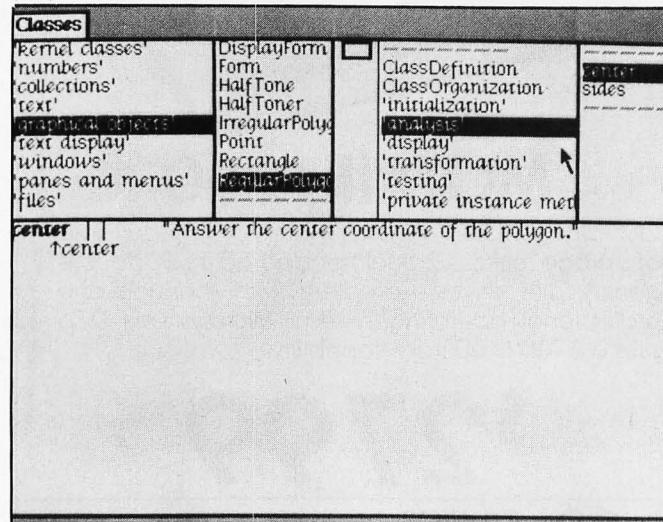
(13a)



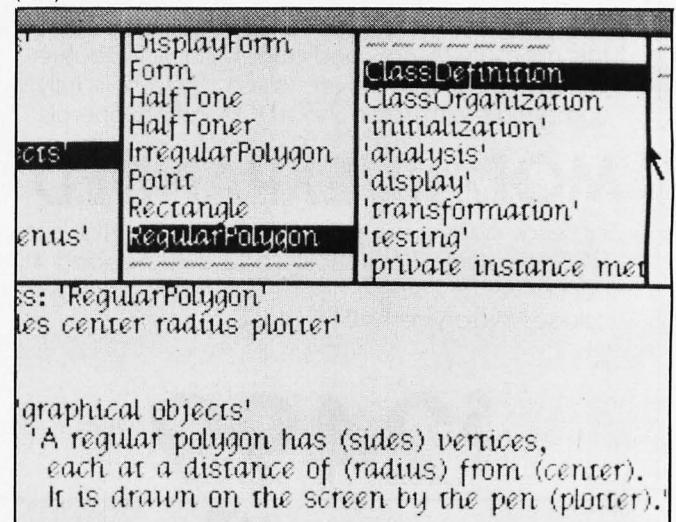
(13b)



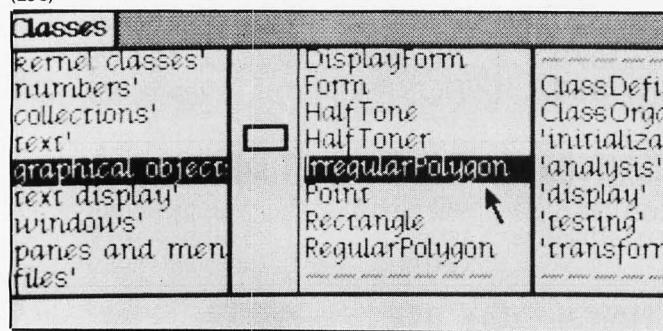
(13c)



(13d)



(13e)



(13f)

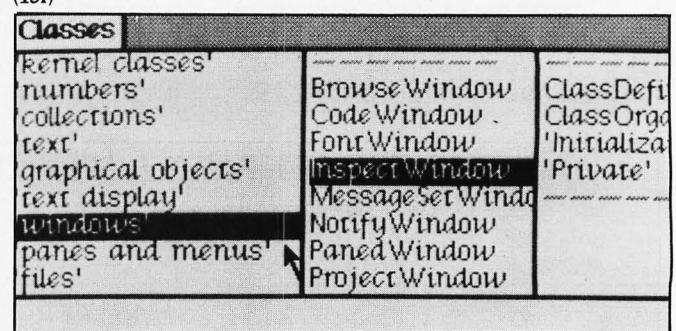


Photo 13: Browsing through existing definitions in Smalltalk.

The following template contains a description of an irregular polygon with the following attributes:

vertices An OrderedCollection of Points.

plotter A pen that can draw an image of the polygon on the screen or on paper.

The following expressions provide an example of creating and using an instance of IrregularPolygon.

triangle ← IrregularPolygon vertices:

(OrderedCollection with: 2 @ 21
with: -25 @ -35
with: 52 @ -7).

triangle translateBy: -90 @ 60.

triangle plot: black.

triangle translateBy: 165 @ 20.

triangle scale: 0.6.

triangle plot: gray. "where gray denotes an ink color"

class name	IrregularPolygon
superclass	Object
instance variable names	vertices plotter
class messages and methods	
<i>initialization</i>	
vertices: aCollection 	"Create an instance of IrregularPolygon whose center is located at the center of the currently active window on the display screen. Screen is a global variable that refers to the hardware display screen." ↑ self new vertices: aCollection center: (Screen activeWindow frame center)
<i>instance messages and methods</i>	
<i>initialization</i>	
vertices: aCollection center: c 	"Initialize all attributes. Class Pen is provided in the system as one way of side effecting the display screen." vertices ← aCollection. plotter ← Pen width: 2. self translateBy: c - self center
<i>analysis</i>	
center sum 	"Answer the center coordinate of the polygon." sum ← 0@0.
vertices do: [:pt sum ← sum + pt].	
sum / self sides	
sides 	"Answer the polygon's number of sides." vertices size
<i>display</i>	
plot: ink 	"Draw an image of the polygon using the specified ink color." plotter penup. plotter goto: vertices last. plotter color: ink. plotter pendn. vertices do: [:pt plotter goto: pt]
	"lift the pen to disable drawing" "position the pen at one vertex" "select the ink color" "lower the pen to enable drawing" "for each vertex" "draw a straight line to it"
<i>transformation</i>	
scale: factor center 	"Scale the polygon by the specified factor." center ← self center.
	"the center of expansion"
vertices ← vertices collect:	"generate new vertex list from old list"
[:pt (pt - center)*factor + center]	
translateBy: deltaXY 	"Change the polygon's location by the specified amount (a Point)" vertices ← vertices collect: [vertex vertex + deltaXY]

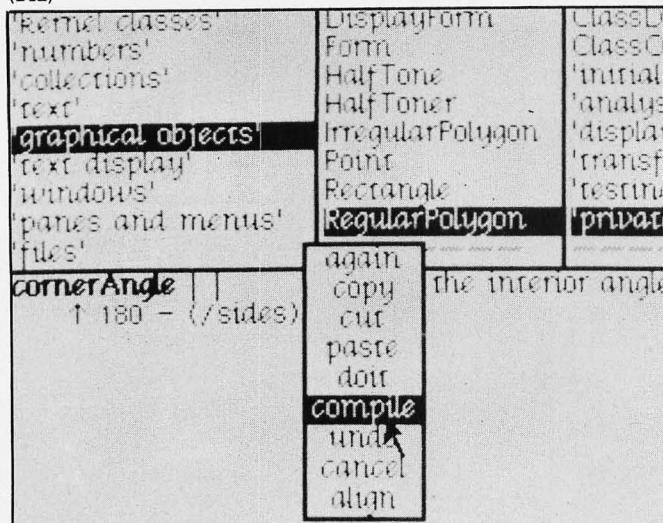
Table 2: Description and class template for class IrregularPolygon.

When you are done editing, pop up the active-pane menu and issue the "compile" command (photo 14a). Compilation takes a few seconds or less because it is *incremental*—that is, you can compile one method at a time. The compiler reports a syntax error to you by inserting a message at the point where the error was

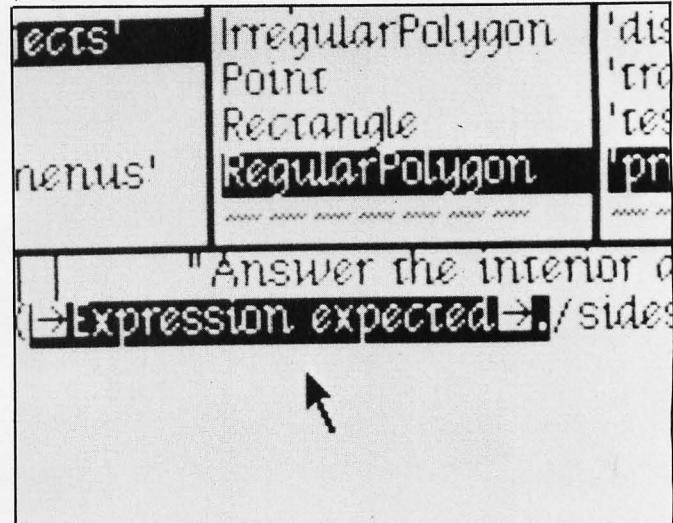
detected and automatically selecting that error message (photo 14b). You can then cut out or overtype the message, make the correction, and immediately reissue the "compile" command.

If you start to revise a definition and change your mind about it, you can pop up the pane menu and issue the

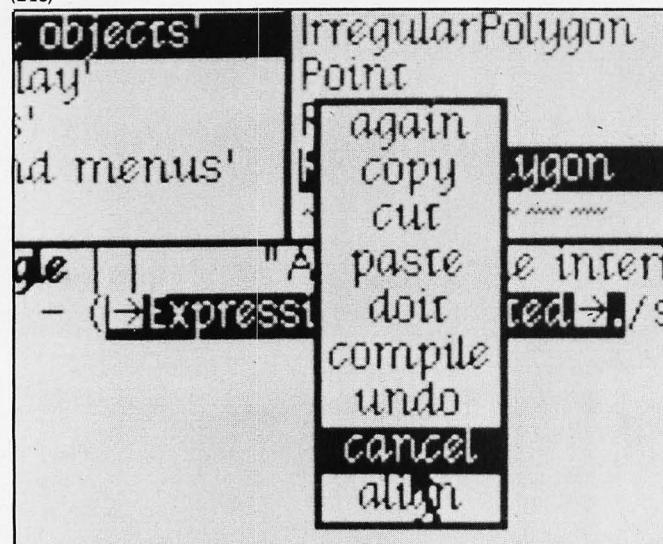
(14a)



(14b)



(14c)



(14d)

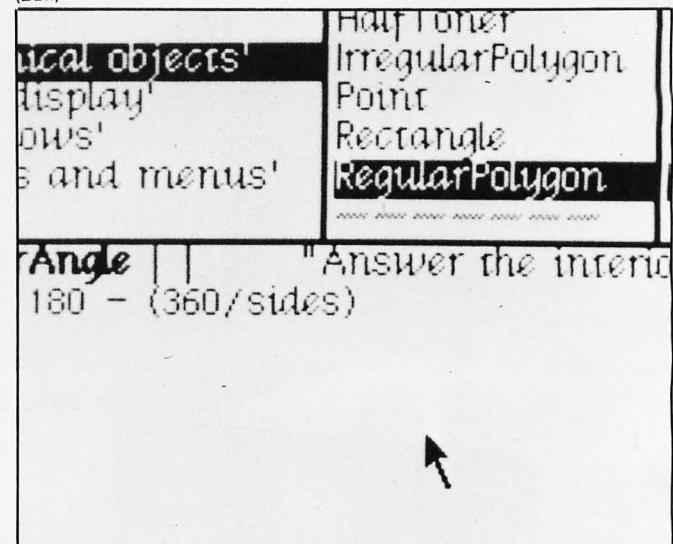


Photo 14: Options during method compilation.

6809 SYSTEMS



6809 SYSTEMS

Featuring the GIMIX mainframe with 30 amp C.V. ferro-resonant power supply; fifteen 50 pin and eight 30 pin slot Mother Board; 2 MHz CPU with time of day clock & battery back-up, 1K RAM, 6840 programmable timer, provisions for 9511A or 9512 Arithmetic processors, and 4 PROM/ROM/RAM sockets that can hold up to 32KB of monitor or user software.

VARIETY: you can have 32KB, 56KB, 128KB and up of static RAM. You can use 5" and/or 8" disk drives, single or double density, single or double sided, and single or double tracking with GIMIX disk controllers. You have a wide choice of serial or parallel I/O cards.

EXPANDABILITY: You can add memory, I/Os, Video or Graphics cards, Arithmetic processors, additional drive capacity, and other hardware now or in the future to this SS50 bus structured system from GIMIX or other SS50 bus compatible manufacturers.

SOFTWARE VERSATILITY: GIMIX systems can use TSC's FLEX or UNIFLEX and MICROWARE'S OS-9 operating systems. A wide variety of software and languages is available for these systems.

QUALITY: All boards are assembled, burned-in, and tested and feature GOLD PLATED BUS CONNECTORS. Only top quality components are used and all boards are fully buffered for maximum system expansion. All boards come complete with bus connectors and all necessary instruction and documentation.

GIMIX designs, manufactures and tests, in-house, their complete line of products. Complete systems are available to fit your needs. Please contact the factory if you have any special requirements.

6809 SYSTEMS



6809 SYSTEMS

For further information, pricing and brochures, contact:



GIMIX INC.
The Company that delivers

Quality Electronic products since 1975.

1337 WEST 37th PLACE, CHICAGO, IL 60609
(312) 927-5510 • TWX 910-221-4055

GIMIX® and GHOST® are registered trademarks of GIMIX Inc.

Flex and Uniflex are trademarks of Technical Systems Consultants Inc. OS9 is a trademark of Microware Inc. See their ads for other GIMIX compatible software.

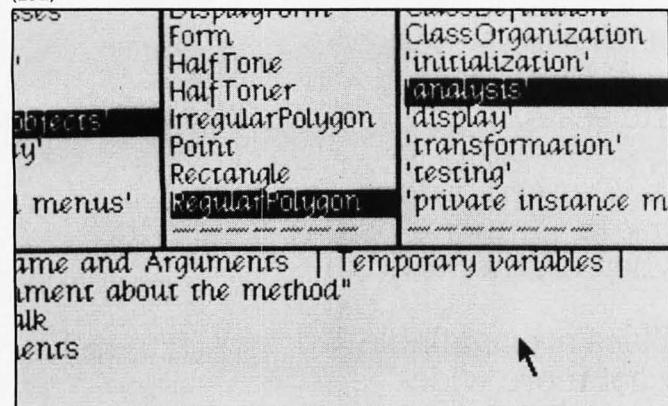
For GIMIX compatible software see Technical Systems Consultants ad page 183 and Microware ad page 112.

"cancel" command (photo 14c). The "cancel" command redisplays the last successfully compiled version of the method (photo 14d). If you cancel by accident, just issue the "undo" command to return the revised version.

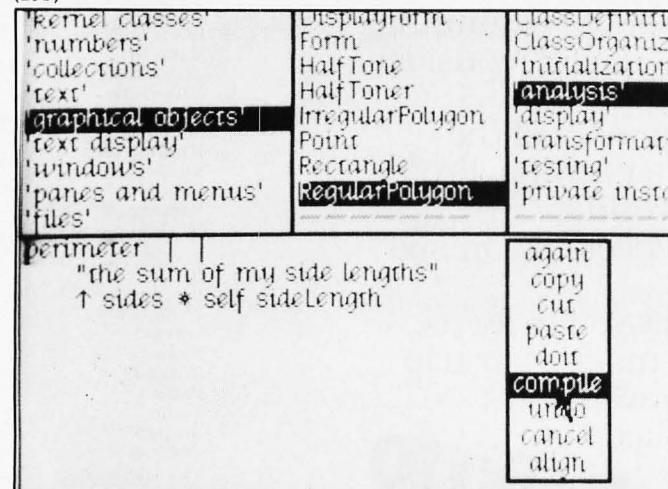
Adding New Definitions

To add a new method definition, select a method category. In the editing pane, a *template* appears for defining a new method (photo 15a). The template reminds you of the required syntax of a method.

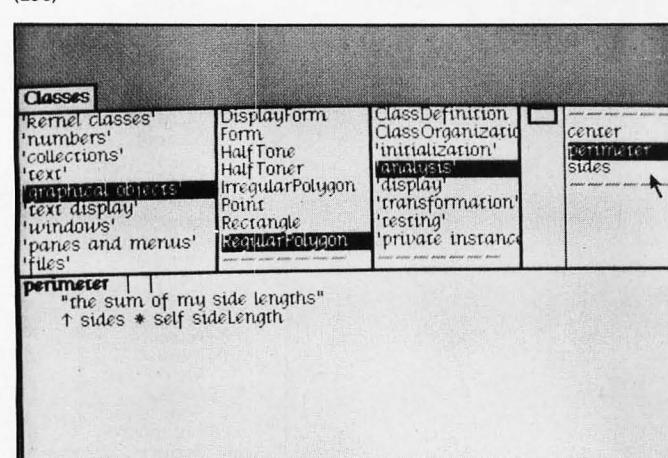
(15a)



(15b)



(15c)



Use standard editing facilities to supply the message pattern, variable list, and body of the method. When the definition is ready, issue the "compile" command (photo 15b).

Once compilation succeeds, the selector of the new method is automatically added to the alphabetized list in the method pane, and the message pattern is automatically changed to boldface in the editing pane (photo 15c).

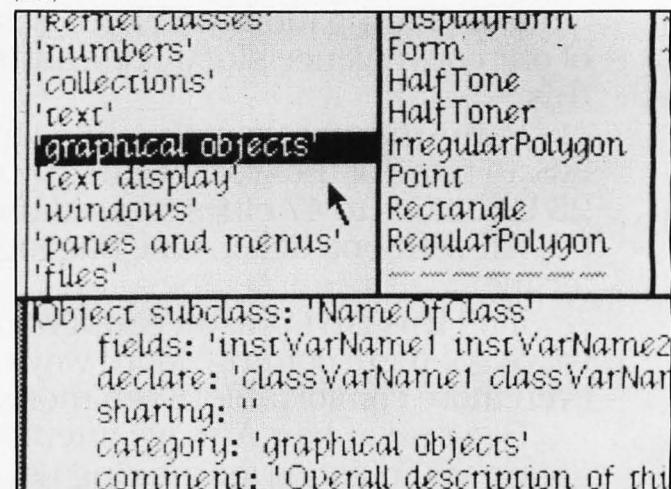
A new class definition is added in an analogous manner. Start by selecting a class category (photo 15d), then fill in a template for defining a new class and compile it (photo 15e). New categories can be added and old categories can be renamed and reorganized.

Program Testing

Let us purposely add a bug to a method and see how it can be tracked down and fixed.

Browse to the method `cornerAngle` in class `RegularPolygon`, cut out the characters "180 —" (photo 16a), and recompile it. In the `RegularPolygon` work-space window, select the test program and issue the "do it" command (photo 16b). Instead of the desired triangle, an open three-sided figure is drawn because of the bug introduced into the angle calculation.

(15d)



(15e)

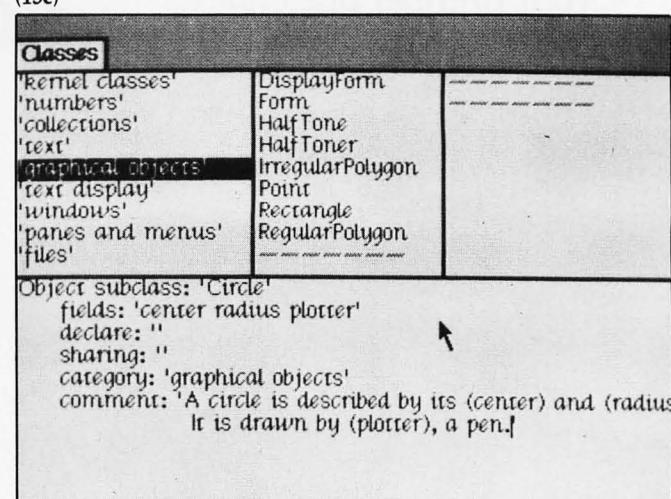


Photo 15: Adding new definitions in Smalltalk.

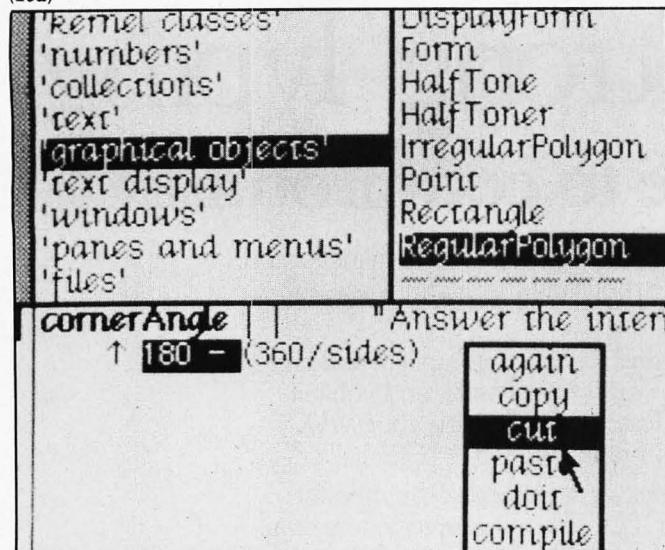
Breakpoints

To track down the bug, let us set a breakpoint in the method cornerAngle. Using standard editing facilities, add the statement:

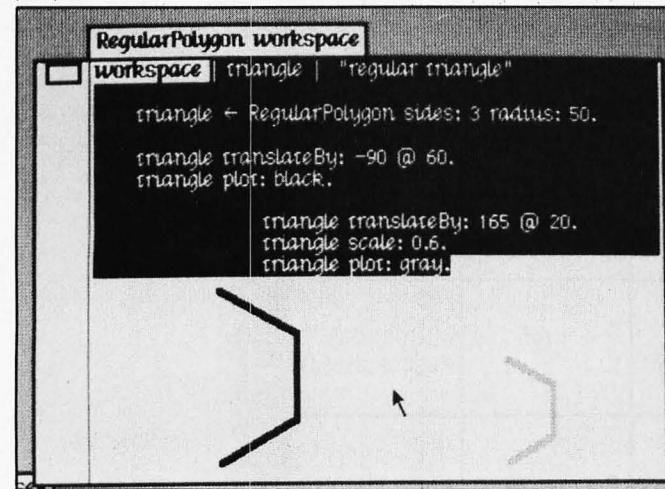
```
self notify: 'about to calculate angle'.
```

before the return statement (photo 16c). Now rerun the

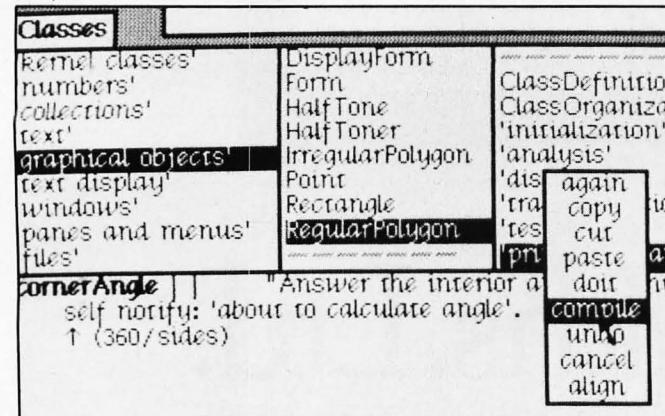
(16a)



(16b)



(16c)



test case. When the computer encounters the breakpoint, a new window appears in midscreen. It is called a "notify window" (photo 16d). The title tab of the notify window says "about to calculate angle".

The notify window has one pane, the *stack pane*. It shows RegularPolygon >>cornerAngle (ie: the class and method in which the breakpoint was encountered). The pop-up menu of that pane offers several commands, including "stack" and "proceed" (photo 16e).

The "proceed" command closes the notify window and continues execution from the breakpoint. If we issue a "proceed" in our example, the same breakpoint will be encountered again immediately because the cornerAngle method is used several times during the execution of the test program.

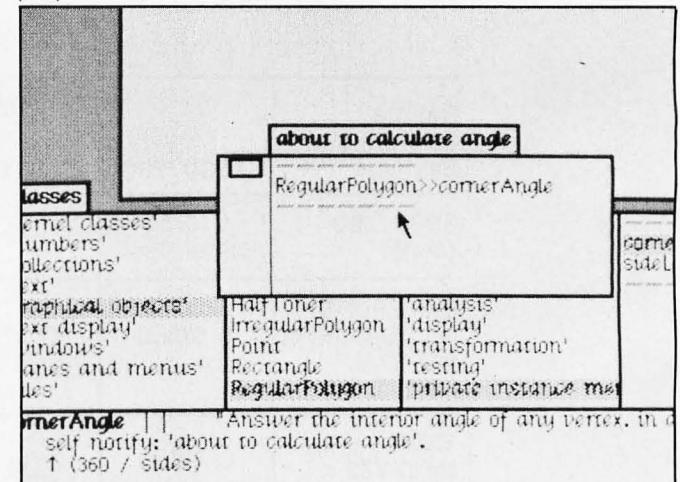
What a Notify Window Can Display

The "stack" command expands the contents of the pane to include messages that have been sent, but have not yet received replies (photo 17a). It reveals that the sender of the message cornerAngle was RegularPolygon >>plot::

The pop-up menu of the notify window offers the usual repertoire, including the "close" and "frame" commands (photo 17b). If "close" were issued, the notify window would disappear from the screen and execution of the

Text continued on page 138

(16d)



(16e)

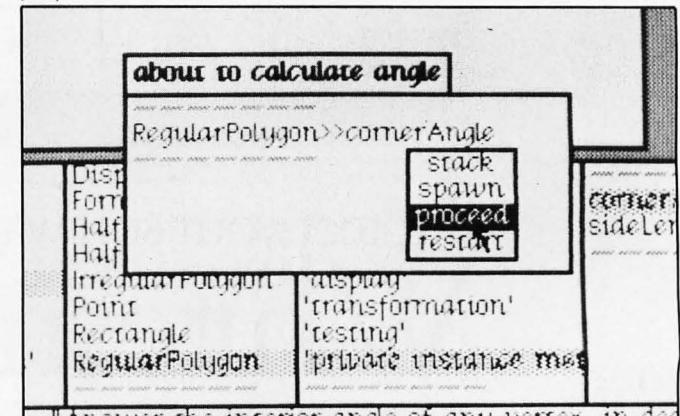


Photo 16: Creating a faulty method for purposes of illustration.

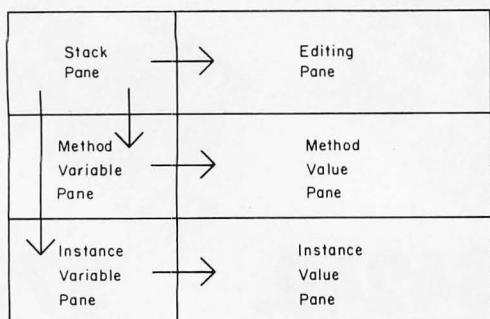


Figure 5: Principal dependencies among panes of a notify window.

Text continued from page 134:

program under test would be aborted. Let us issue the "frame" command instead. The notify window grows larger and acquires a total of six panes (photo 17c). Their interdependencies are diagrammed in figure 5.

The upper left pane is the stack pane retained from before. The upper right pane is an editing pane. If you select RegularPolygon >> plot: in the stack pane, its method definition appears in the editing pane. You can scroll through the definition and even edit it there and recompile as in the browser.

The middle two panes are the "context variable" and

"context value" panes. They are analogous to the two panes of an inspect window, but, in this case, the variables you can examine are the arguments and local variables of the method selected in the stack pane. Click ink in the variable pane to see its value in the value pane.

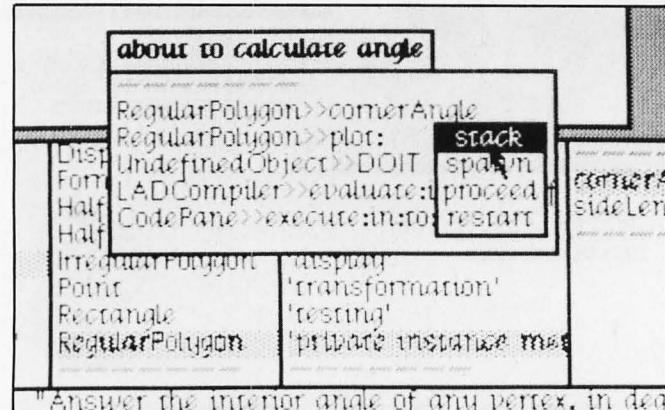
The bottom two panes are the "instance variable" and "instance value" panes. They also are analogous to the panes of an inspect window. They let you examine the instance variables of the receiver of the message selected in the stack pane. Click center to see its value appear in the value pane.

You can type statements into the value panes and execute them using "do it" (photo 17d). They will be executed in the context of the method selected in the stack pane—that is, they may refer to arguments and local variables of the method and to instance variables.

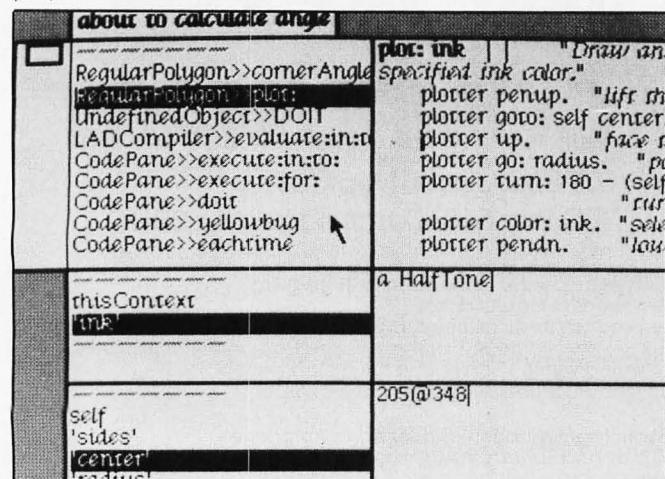
Debugging

You could step through the execution of the method in the editing pane. You would select one statement at a time in the editing pane and issue the "do it" command. To close in on the planted bug, we can evaluate self.cornerAngle, an expression on the last line of the method. Select that expression and issue the "do it" command (photo 18a). The answer, 120, appears to the right of the question (photo 18b). Since the interior angle of a regular

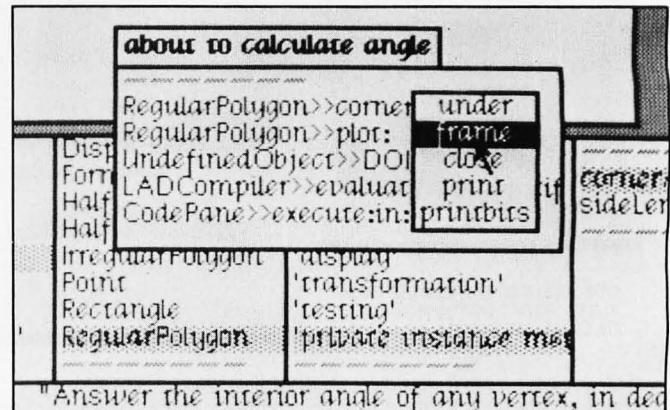
(17a)



(17c)



(17b)



(17d)

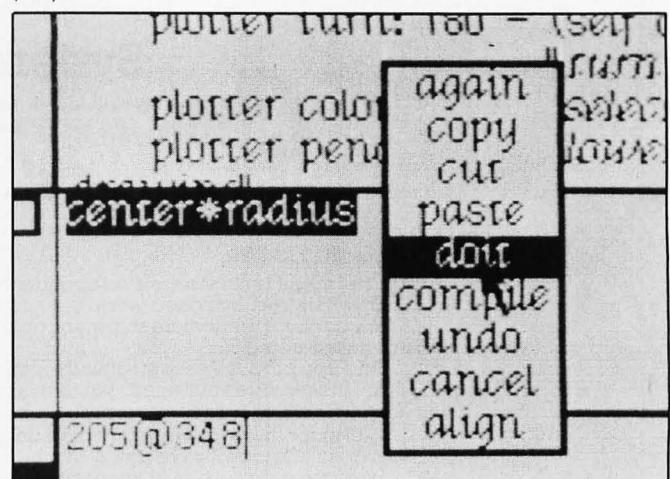


Photo 17: Use of the "notify" window.

(18a)

```

again          "Draw an image of the polygon
copy           "lift the pen so disable
cut            "position the pen at center. "position the pen
paste          "face the top of the screen
doit           "position it as a corner
compile        0 - (self cornerAngle/2).
undo           "turn to face along a side
cancel          "select the ink color
align           "lower the pen so enable

```

radius 10250@17400

(18b)

"Draw an image of the polygon using
ink color."
up. "lift the pen so disable drawing
: self center." position the pen center
"face the top of the screen"
radius. "position it as a corner"
t: 180 - (self cornerAngle 120/2).
"turn to face along a side"
r: ink. "select the ink color"
dn. "lower the pen so enable

10250@17400

(18c)

about to calculate angle

RegularPolygon >>cornerAngle	cornerAngle "Answer vertex, in degrees" self notify: 'about to calculate' ↑ (360/sides)
RegularPolygon >>plot:	
UndefinedObject >>DOIT	
LADCompiler >>evaluate:in:to:	
CodePane >>execute:in:to:	
CodePane >>execute:for:	
CodePane >>doit	
CodePane >>yellowbug	
CodePane >>eachtime	
CodePane(Object) >>startup	

thisContext

RegularPolygon >>cornerAngle

(18d)

about to calculate angle

RegularPolygon >>cornerAngle	cornerAngle "Answer vertex, in degrees" self notify: 'about to calculate' ↑ (360/sides)
RegularPolygon >>plot:	
UndefinedObject >>DOIT	
LADCompiler >>evaluate	
CodePane >>execute:in:to:	
CodePane >>execute:for:	
CodePane >>doit	
CodePane >>yellowbug	
CodePane >>eachtime	

thisContext

RegularPolygon >>cornerAngle

triangle is 60 degrees, we have found the planted bug.

Now select RegularPolygon>>cornerAngle in the stack pane. Its method definition, including the breakpoint we set, appears in the editing pane (photo 18c). Use standard editing to remove the breakpoint, correct the error, and recompile the editing pane (photo 18d).

You can randomly access any level in the stack by clicking it in the stack pane.

Resumption

After recompiling a method, you can resume execution from the beginning of any method on the stack using the "restart" command in the stack-pane menu (photo 19a). This lets the test proceed (photo 19b) without having to start over from the work-space window. Resumption of execution after a correction is a handy capability when a program that has been running well encounters a minor bug.

The entire stack of the process under test was saved in the notify window. When a notify window appears, the rest of the system is not preempted. You are not required to deal with the notify window when it appears. You can

(19a)

about to calculate angle

RegularPolygon >>cornerAngle	cornerAngle "Answer vertex, in degrees"
RegularPolygon >>plot:	vertex, in degrees" ↑ 180 - (360/sides)
UndefinedObject >>DOIT	
LADCompiler >>evaluate:in:to:	
CodePane >>execute	stack
CodePane >>execute	spawn
CodePane >>doit	proceed
CodePane >>yellowbug	restart
CodePane >>eachtime	

thisContext

RegularPolygon >>cornerAngle

(19b)

RegularPolygon workspace

workspace | triangle | "regular triangle"

```

triangle ← RegularPolygon sides: 3 radius: 50.
triangle translateBy: -90 @ 60.
triangle plot: black.

triangle translateBy: 165 @ 20.
triangle scale: 0.6.
triangle plot: gray.

```

Photo 19: Compilation of a faulty method can be continued without restarting, once the error has been corrected.

Photo 18: Debugging a faulty method.

work in other windows and come back to it later, cause other notify windows to be created, or work a little in the notify window and then do something else. There are no modes.

Error Notifications

Error messages are no different from breakpoints, ex-

cept that if they are supposed to be "unrecoverable" they are programmed as:

self error: 'error whatever'.

If the user "proceeds" out of the notify window after an error, the process under test is terminated.

The most frequently encountered Smalltalk error is "Message not understood." It occurs when a method is sent to an object and neither that object's class nor any of its superclasses defines a method to receive that message. Let us edit the method sideLength (photo 20a) to send the message cosine instead of cos. After recompiling that method (photo 20b) and reexecuting the test program, a notify window appears (photo 20c) to announce that class Real and its superclasses do not define cosine.

In most programming systems, equivalent error conditions such as "undeclared procedure" and "wrong number of arguments" are issued at compile time. Smalltalk cannot detect these conditions until run time because variables are not declared as to type. At run time, the object sent the message cosine could be an instance of a class that did define a method of that name.

Type Checking

When we program in languages like Pascal, we depend on type checking to catch procedure-call errors early in the software-development process. In return, we have to take extra time maintaining type declarations, and we lose the very powerful ability to define "generic" or "polymorphic" procedures with the same name but with parameters of varying types.

Type checking is important in most systems for four reasons, none of which is very important in Smalltalk:

- Without type checking, a program in most languages can "crash" in mysterious ways at run time. Even with type checking, most programming systems can crash due to uninitialized variables, dangling references, etc. Languages with this feature are sometimes called "unsafe." Examples of unsafe languages are Pascal, PL/1, and C. Examples of fairly safe languages are BASIC and LISP. Smalltalk is a safe language. It cannot be wiped out by normal programming. In particular, it never crashes when there are "type mismatches." It just reports a "Message not understood" error and helps the programmer quickly find and fix the problem through the notify window.
- In most systems, the edit-compile-debug cycle is so tedious that early error detection is indispensable. In Smalltalk, type errors are found early in testing, along with value-range errors and other bugs.
- Type declarations help to document programs. This is true, but well-chosen variable names and pertinent comments provide more specific information than do type declarations. A poor documenter can convey as little information in a strongly typed program as in an untyped program.

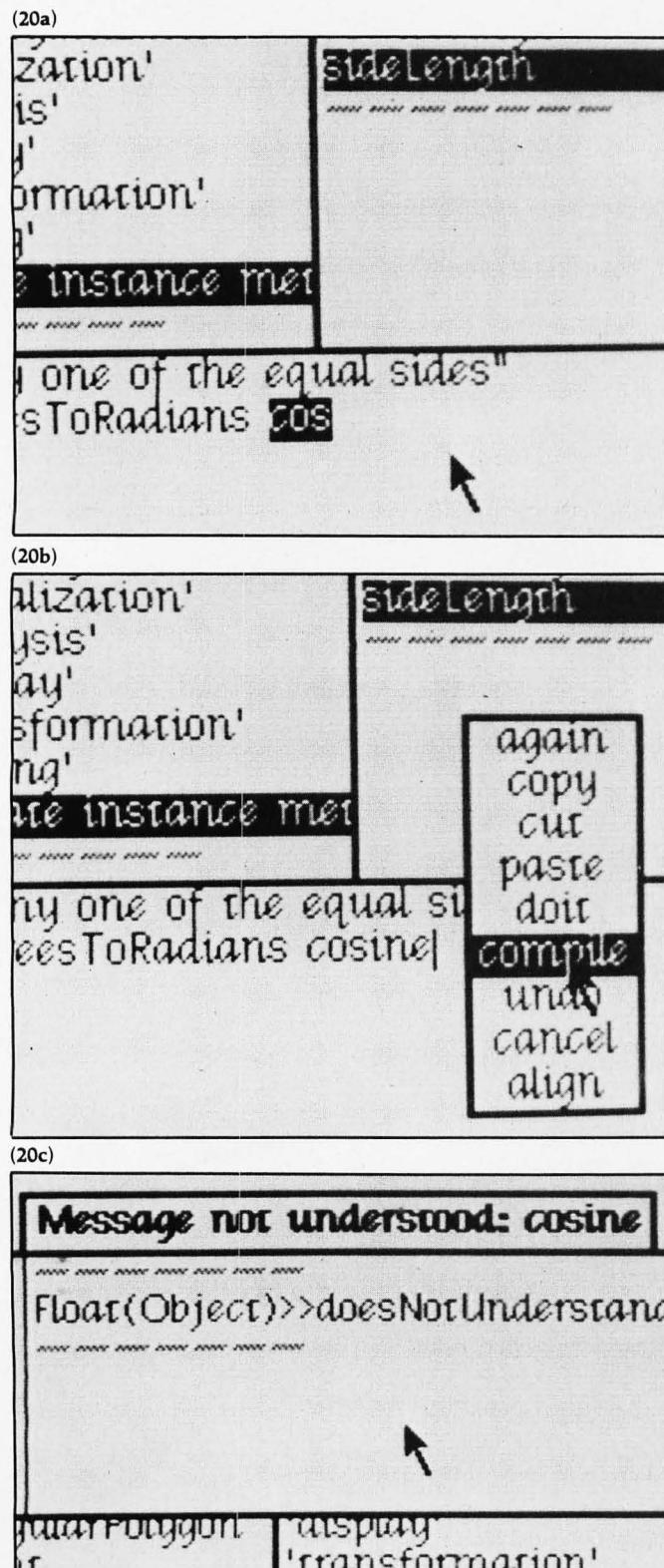


Photo 20: Displaying an error in a faulty method.

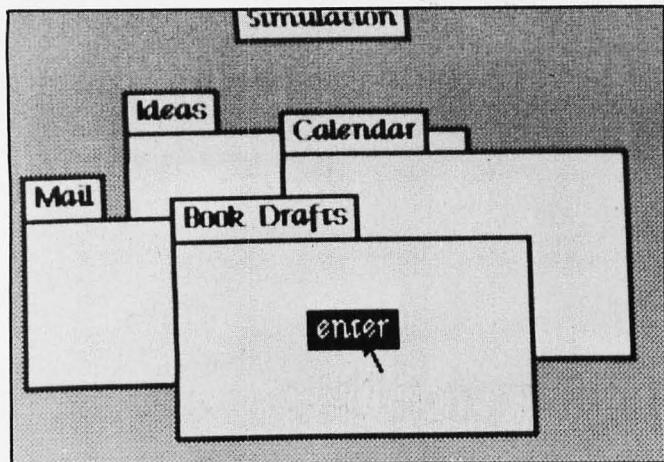
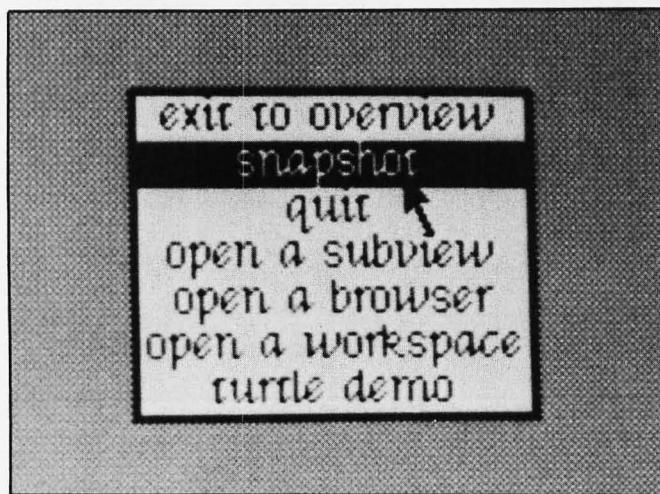


Photo 21: Project windows in Smalltalk. Each window, when selected, makes available all the windows associated with that project.

(22a)



(22b)

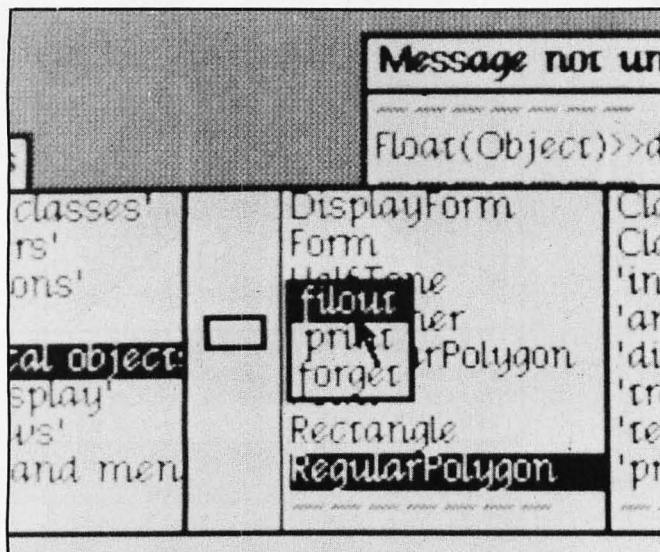


Photo 22: Recording results in Smalltalk. The current state of the Smalltalk system can be saved with "snapshot." Smalltalk code can be saved to a text file by using "filout" and restored by using "filin."

- Most compilers can generate more efficient object code if types are declared. Existing implementations of Smalltalk cannot take advantage of type declarations. We expect that future versions will have that ability. At that time, type declarations may be added to the language. They probably will be supplied by the system rather than the user, using a program-analysis technique called "type inference."

Project Windows

Although overlapping windows enable you to keep the state of several tasks on the screen at the same time, you may sometimes be working on several entirely different *projects*, each involving several tasks. Smalltalk lets you have a different "desk top" for each project. On each desk top are windows for the tasks involved in that project. To help you travel from one desk top to another, a desk top can have one or more *project windows* that show you other available desk tops and let you switch to one of them (photo 21).

Saving Programs

In unintegrated systems, you create a program using standard text-editing facilities. Then, using standard utility programs, you can obtain a program listing on paper, back up the program on other media, and transmit the program to other people. In an integrated system, equivalent capabilities must be provided within the system itself. Some of the program-saving capabilities of Smalltalk are described briefly below.

One important facility is the *snapshot* (photo 22a). The entire state of the Smalltalk environment—including class and method definitions, data objects, suspended processes, windows on the screen, and project desk tops—can be momentarily frozen and saved on secondary storage. The snapshot can be restored later and resumed. People familiar with the *sysout* in InterLISP or the *workspace* concept in APL will understand the benefit of this facility.

Another facility allows definitions of one or more methods or classes to be listed on a printer. A related facility is *filin/filout*. The *filout* message (photo 22b) writes an ASCII representation of one or more definitions onto a conventional text file. The definitions can then be transfused into another Smalltalk environment by using the *filin* message in that environment.

Often, during a programming session, the user changes a number of method definitions that are scattered throughout many classes and cannot recall which ones were changed. The *changes* facility automatically keeps a record of what definitions changed in each project, and makes it easy for the user to *filout* those definitions at the end of the session.

Implementation of the Environment

Because Smalltalk is an integrated environment, all the facilities described in this article are implemented in the

Text continued on page 147

Text continued from page 144:

high-level language, including modeless editing, windows, the compiler, and the notify mechanism. This was possible because Smalltalk represents everything, including the dynamic state of its own processes, as objects that remember their own state and that can be sent messages by other objects. Using the browser, you can examine and (carefully) change the definitions of the software-development aids.

In the implementation of Smalltalk-76, classes *InspectWindow*, *BrowseWindow*, and *NotifyWindow* are all tiny subclasses of class *PanedWindow*, which defines their common behavior. Similarly, classes *StackPane*, *VariablePane*, *ValuePane*, and so on, are all tiny subclasses of class *ListPane*. The superclass defines common behavior such as scrolling and selecting entries.

If someone shows you a system claimed to be "Smalltalk," find out whether the software-development aids exist and whether they are programmed as class definitions in the high-level language. If not, the system is not bona fide.

Conclusions

The Smalltalk programming environment is *reactive*. That is, the user tells it what to do and it reacts, instead of the other way around. To enable the user to switch between tasks, the state of the tasks is preserved in instantly accessible windows that overlap on desk tops. To give the user the maximum freedom of choice at every moment, modes rarely occur in the user interface. The result of this organization is that tasks, including software-development tasks, can be accomplished with greater speed and less frustration than is usually encountered in computer systems. ■

References

1. Sneiringer, J. "User-Interface Design for Text Editing: A Case Study." *Software—Practice and Experience* 8, pages 543 thru 557, 1978.
2. Swinehart, D C (thesis). "Copilot: A Multiple Process Approach to Interactive Programming Systems." *Stanford Artificial Intelligence Laboratory Memo AIM-230*, Stanford University, July 1974.
3. Kay, A and A Goldberg. "Personal Dynamic Media." *Computer*, March 1977 (originally published as *Xerox PARC Technical Report SSL-76-1*, March 1976, out of print).
4. English, W, D Engelbart, and M Berman. "Display-Selection Techniques for Text Manipulation." *IEEE Transactions on Human Factors in Electronics*, volume 8, number 1, pages 21 thru 31, 1977.
5. Card, S, T Moran, and A Newell. "The Keystroke-Level Model for User Performance Time with Interactive Systems." *Communications of the ACM*, volume 23, number 7, July 1980.
6. Goldberg, A and D Robson. "A Metaphor for User-Interface Design." *Proceedings of the Twelfth Hawaii International Conference on System Sciences*, volume 6, number 1, pages 148 thru 157, 1979.
7. Borning, A. "ThingLab—A Constraint-Oriented Simulation Laboratory." To appear in *ACM Transactions on Programming Languages and Systems* (originally published as *Stanford Computer Science Report STAN-CS-79-746* and *Xerox PARC Technical Report SSL-79-3*, July 1979, out of print).

INNOVATIVE PRODUCTS

COMPLETE CATALOG OF COMPUTER EQUIPMENT AT DISCOUNT PRICES.

65K S-100 DYNAMIC MEMORY BY FAMOUS NAME MANUFACTURER

unbelievable price:

\$299

FEATURES: Fully Compatible with IEEE-696. Assembled, tested and fully populated with 65K of 200 ns. brand name memory. Bank selectable, 4Mhz. operation, extended addressing, invisible refresh 8/16 bit operation (per IEEE-696) and many other features. One year warranty, manual included... Hurry, limited quantities!

TELEVIDEO TERMINALS:

	TELEVIDEO TERMINALS:	EPSON PRINTERS:
Model 912C	695.00	Model MX-70
Model 920C	720.00	Model MX-80
New! Model 950	995.00	\$CALL

EPSON PRINTERS:

Q.T. COMPUTER SYSTEMS:

MODEL	DESCRIPTION (ALL ASSEMBLED & TESTED)	PRICE
MF+DD12A	25 A. Mainframe: 12 Slot, S-100, Dual 8" Drive	\$625.00
MF+18A	25 A. Tabletop Mainframe: 18 Slot, S-100	450.00
MF+12A	25 A. Mainframe: 12 Slot, S-100, Dual 5" Drive	450.00
Z-80A	Single Board, Z-80 Computer 2/4 MHz. S-100	185.00
Z-80A/2.2	Monitor Program for Z-80A & Tarbell or VF-1	30.00
RAM 32	32K Static RAM Board, Low Power, S-100	499.00
QTClOA	2 Serial & 4 Parallel Port I/O, S-100	339.00
QTCCSA	Clock Calendar Board, S-100	125.00
QTCCAA	Clock Calendar Board, for Apple	125.00
QTCCTRS80	Clock Calendar Module for TRS-80	125.00

VISTA COMPUTER COMPANY:

MODEL	DESCRIPTION (ALL ASSEMBLED & TESTED)	PRICE
V300/25	Letter Quality Printer, 25 CPS	\$1895.00
V300/45	Letter Quality Printer, 45 CPS	2195.00
V-DRIVE	Dual 8" Disk Cabinet, Power Supply, Rack Mountable	375.00

COMPUTER SYSTEM RESOURCES:

MODEL	DESCRIPTION (ALL ASSEMBLED & TESTED)	PRICE
RAM 16	16K Static RAM, S-100	\$189.00
RAM 16C	Low Power (CMOS) Version of RAM 16, 1.3 Watt	259.00
RAM 65	16K Static RAM, I/O Port Bank Select, S-100	199.00
RAM 65C	Low Power (CMOS) Version of RAM 65, 1.4 Watt	279.00
RAM 32	32K Low Power 8/16 Bit Static RAM, Fully IEEE 696	499.00
RAM 256	256K 8/16 Bit Dynamic RAM, Fully IEEE 696	1295.00
I/O-8	8 Port Serial I/O, Fully IEEE, up to 200K BAUD	395.00

ORDERING INFORMATION:

Minimum Order is \$15.00. Prices quoted do not include shipping and handling. Foreign orders require prepayment by MICR or Money Order in U.S. funds. Purchase Orders accepted from U.S. Government & firms with published A1 Rating from Dunn & Bradstreet. All other orders require prepayment, charge card, or COD shipment.

TOLL-FREE PHONE NUMBER:

Call Toll-Free: 1-800-555-1212, and ask the operator for our New, Toll-Free (800) Number. In Alaska, California, and Hawaii, call us collect at (213) 883-3244.

TO RECEIVE OUR FREE CATALOG:

Call our Toll-Free (800) number, or circle Reader Service #.

7131 OWENSMOUTH AVE. / #21D
CANOGA PARK, CALIFORNIA 91303
213 / 883-3244

INNOVATIVE
PRODUCTS