Last Edited by: Jack Kent May 8, 1987 1:46:52 pm PDT

The Briefing Blurb:

Exploring the Ethernet with Mouse and Keyboard

1987 Edition

Lyle Ramshaw (June 7, 1983)

An immigration document in the tradition of Roy Levin's *A Field Guide to Alto-Land.*

Filed on: [Cedar]<CedarChest®>Documentation>BriefingBlurb.tioga

© Copyright 1984, 1985, 1986, 1987 Xerox Corporation. All rights reserved.

Abstract: This document is a general introduction to the computing environment at PARC slanted towards the needs and interests of newcomers to the Computer Science Laboratory. If you are looking at this document on-line from within the editor named Tioga, you might want to use the level-clipping functions to see the overall structure rather than simply plowing straight through. Click the "Levels" button in the top menu, then click "FirstLevelOnly" in the new menu that appears. That will show you the major section headings. Click "MoreLevels" to see the subsections, or click "AllLevels" to read the details.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

For Internal Xerox Use Only

Raison d'Etre

The purpose of this document is to help immigrants adapt to the local computing community. By "the local community", I mean primarily the Computer Science Lab of the Xerox Palo Alto Research Center, better known by the acronym CSL. Immigrants to other computing communities within Xerox may also find this document of interest, but I make no guarantees. I shall assume herein that said immigrants know quite a bit about computer science in general. Hence, I shall concentrate upon discussing the idiosyncratic characteristics of the local hardware environment, software environment, social environment, linguistic environment, and the like.

You will doubtless read many documents while you are at Xerox. A common convention observed in many manuals and memos is that fine points or items of complex technical content peripheral to the main discussion appear in small type, like this paragraph. You will soon discover that you cannot resist reading this fine print and that, despite its diminutive stature, it draws your eyes like a magnet. This document has such passages as well, just so that you can begin to enjoy ferreting out the diamonds hidden in the mountain of coal.

There is a great deal of useful information available on-line at Xerox in the form of documents and source programs. Reading them is often very helpful, but finding them can be a nuisance. Throughout this document, references to on-line material are indicated by <*n*>, where *n* is a citation number in the bibliography at the end of this document. Standard citations to the open literature appear as [*n*].

If you are fortunate enough to be reading this document from within Tioga (the Cedar editor), you should pause at this point to try out the "Def" command. If you were to select the three characters "<n>" in the preceding paragraph and then click the "Def" command with the middle mouse button, you would then find yourself looking at the place in this document where "<n>" is defined, that is, where it appears followed by a colon. You could then get back to this section of the document by clicking the "PrevPlace" command with any mouse button. The "Def" command is almost as good as an automatic indexing facility. On another topic, you might try clicking the "FirstLevelOnly" button (click "Levels" first if you can't find the "FirstLevelOnly" button), and then clicking "MoreLevels" a few times. Try scrolling a bit too. The Tioga "levels" commands are almost as good as an automatic table of contents.

Reading a document from front to back can be mighty boring. Fortunately, this document is so disorganized that it is not at all clear that it really has a front and a back in any normal sense. You might as well just browse through and read the parts that look interesting. To help out the browsers in my reading community, I have more or less abandoned the custom of being careful to define my terms before I use them. Instead, all the relevant terms, acronyms, and the like have been collected in a separate Glossary. Some information is contained *only* in the Glossary, so you may want to skim through it later (or now, for that matter). The "Def" command in Tioga is particularly helpful when browsing the Glossary from within Cedar: try selecting the word "Tioga", and then clicking the "Def" button in the Glossary viewer, for example. While writing the Glossary, I assumed that you have a basic knowledge of computer science, and a modicum of common sense: don't expect to find terms like "computer" and "network" in the Glossary.

A caveat: this document contains a fair amount of archaic information about the glorious Alto and its software environment. Although this information provides an interesting historical perspective on PARC computing facilities, it is currently of little practical value to Cedar users. Paragraphs consisting entirely of such information appear in small type and are prefaced by "*History:*" so that you are sure to read them.

Naming Things

At the outset, you should know something about the names of the creatures that you will find here. The prevailing local philosophy about naming systems is perhaps somewhat different from the trend elsewhere. We do have our share of alphabet soup, that is, systems and languages that are named by acronyms of varying degrees of cuteness and artificiality; consider, for example: PARC, FTP, IFS. But we are trying to avoid making this situation any worse. To this worthy end, names for hardware and software systems are frequently taken from the *Sunset Western Garden Book* [1]; Grapevine servers are named after wines; Dorados are named after capital ships; Pilot releases are named after California rivers. As this convention about names does not meet with universal approval, it seems inappropriate to offer a justification of the underlying philosophy without offering equal time to the opposition. You will doubtless provoke a far more interesting discussion if you advance your own views on naming to almost anyone wandering in the corridors.

While we are on the general topic of the names of things, we should discuss for a moment the local customs for constructing single identifiers out of multiple word phrases. Suppose that you would like to name a variable in your program "name several words long". In some environments, a special character that isn't a letter but that acts something like a letter is used as a word separator within identifiers; this leads to names such as

"name!several!words!long" or "name_several_words_long".

No such character is in common use locally, however. Instead, shifting between upper and lower case is used to show the word boundaries, leading to the name

"NameSeveralWordsLong".

Some people, including Don Knuth, think that identifiers with mixed case look terribly ugly. I refuse to get sucked into expressing my opinion in this document; once again, I exhort you to espouse your views in the corridors.

There are several fine points that I should mention as well. As a general rule, case is significant for identifiers in the local programming languages, but case is not significant in file names or in Grapevine R-names. Thus, the Cedar identifiers "REF", "Ref", and "ref" are quite distinct, but the file names "BriefingBlurb.tioga" and "briefingblurb.tioga" are equivalent, as are the R-names "Ramshaw.PA" and "ramshaw.pa". In Mesa and Cedar, there is a further convention that the case of the first letter of an identifier is used to distinguish fancy objects, such as procedures and types, from simple ones, such as integers and reals. Thus, the identifier name "ProcWithFiveWordName" begins with an upper case "P", but the name "integerWithFiveWordName" begins with a lower case "i". The latter form looks very strange to most people when they first see it. When you first tasted an olive, you probably didn't like it. Now, you probably do. Give these capitalization conventions the same chance that you would an olive.

These capitalization conventions don't work too well when acronyms and normal words appear together in one identifier. Suppose, for example, that I wanted to introduce an identifier named "FTP version number". Logic would demand "FTPVersionNumber", but this doesn't look quite right: many people would be probably write "FTPversionNumber" instead. Of course, since a version number is probably an integer, it should really be "fTPVersionNumber". Ugh. Perhaps case is being used for too many purposes?

Local Hardware

Most of the offices and some of the alcoves around PARC have personal computers in them of one flavor or another. The first of these was the Alto. There are more than a thousand Altos in existence now, spread throughout Xerox, the four universities in the University Grant program (U. of Rochester, CMU, MIT, and Stanford), and other places. By now, almost of the local personal Altos have been replaced by various flavors of D-machines: Dorados, Dandelions, and (rarely) Dolphins. You will still find a number of Altos in use as server machines. Both D-machines and Altos come equipped with bitmap displays, mice, and Ethernet interfaces. Let's discuss these components first, and then turn our attention to the various personal computers that contain them.

Bitmap Displays

First, let's talk about displays. Different displays use different representations of images. A character display represents its image as a sequence of character codes. This is a very compact representation, but not a very flexible one; text is all you can get, and probably in only a limited selection of fonts. A vector display represents its image as a list of vector coordinates. This works very well for certain varieties of line drawings, but not so well for filled areas or text. A bitmap display, on the other hand, produces an image by taking a large matrix of zeros and ones, and putting white where the zeros are and black where the ones are (or vice versa). The great advantage of bitmap displays are their flexibility: you can specify a tremendous number of images by giving even a relatively small array of bits. Cursors and icons are two large classes of prominent examples. Of course, you do have to supply enough memory to hold all those bits. Altos and D-machines store their bitmaps in main storage. An alternative would be to provide a special chunk of memory on the side where the display's image sits; such a memory is often called a *frame buffer*.

The primary display of the Alto is a bitmap that is 608 pixels wide by 808 pixels high. Such a display is almost large enough to do a reasonable job of rendering a single 8.5" by 11" page of text. The CRT on a D-machine has the long axis horizontal instead of vertical, giving a bitmap display that is 1024 pixels wide by 808 high. It had to be 808 high so that D-machines could emulate Altos, of course. The extra space allows you to have something else on the screen as well as the somewhat scrunched page of text that you are editing.

Ere I leave you with a mistaken impression, let me note in passing that bitmap displays are not the final solution to all of the world's problems. Raster displays that can produce various levels of gray as well as black and white can depict images free of the "jaggies" and other artifacts that are inherent in bitmap displays [2]. And, for some purposes, color is well worth its substantial expense. If you see a second CRT standing next to the bitmap display, it is probably a color display connected to the same Dorado. Most of the Imaging folks and VLSI designers have this extra bit of equipment, as do a few fortunate Systems people.

Mice

But now on to mice. A mouse has two obvious properties—it rolls and it clicks. Inside the machine, the mouse position and the display cursor position are completely unrelated; but most software arranges for the cursor to "track" the mouse's movements. The three mouse buttons go by various names; "left", "middle", and "right" is one set of names. The mouse buttons are also called "red", "yellow", and "blue" respectively, even though physically they are nearly always black. These colorful names were proposed at an earlier time when some of the mice had their buttons running horizontally instead of vertically. Using colors (even imaginary ones!) worked better than switching back and forth between the nomenclatures "top-middle-bottom" and "left-

middle-right".

Mice also come in two basic flavors: mechanical and optical. Our current mechanical mice roll on three balls: two small ones, and one large one. Motion of the large ball is sensed by two little wipers inside the mouse, one sensing side to side rolling while the other senses forward and backward rolling. The motion of each wiper drives a commutator, and little feelers slide along the commutator, producing the electrical signals that the listening computer can decode. Building one of these little gadgets is not quite as hard as building a Swiss watch, but it's in the same league. The optical mice are a more recent innovation. An optical mouse lives on a special pad, covered with little white dots on a black background. A lens in the mouse images a portion of the pad onto the surface of a custom integrated circuit. This IC has sixteen light-sensitive regions, some of which notice that they are being shined on by the image of a white dot on the pad. As the mouse slides along the pad on its Teflon-coated underbelly, the images of the white dots move across the IC; it is subtly constructed so as to observe this phenomenon, and take appropriate electrical action. For more details on this interesting application of a custom chip, you might enjoy checking out Dick Lyon's blue-and-white report on the subject [3].

Ethernets and Internets

Two's company, three's a network. A collection of machines within reasonable proximity is hooked together by an Ethernet; if that doesn't sound familiar, I know of some blue-and-whites that you might like to browse [4,5]. Ethernets are connected to each other by Gateways and phone lines, which for most purposes allow us to ignore the topology of the resulting network. The resulting network as a whole is called an *Internet*. Occasionally, it's nice to know where things *really* are; generally, a map of the internet is posted on some wall in CSL.

Ethernets come in two flavors: old and new. The old one runs at 3 MBits/sec, and should now be referred to as the "Experimental Ethernet". The unqualified name "Ethernet" should be reserved for the new one, the standardized version used in OSD products: it runs at 10 MBits/sec. Within PARC, there are 8 Ethernets. For starters, each of the three floors contains both a 3MBit net and a 10MBit net. The second floor 3MBit net got too long, so it was split into two. The 8th is a 1.5MBit Ethernet used by the voice project. (The Etherphones use a chip made for the copiers. Did you know that many copiers have an Ethernet in them?)

We all know how uncommunicative computers can be when left to their own devices. That's why we invent careful protocols for them to use in talking to each other. There are two entire worlds of protocols that are spoken on our various Ethernets as well: old and new. The old ones are called PUP-based (PARC Universal Packet) [7] and are used exclusively within PARC. The new ones, known by the acronym NS (Network Systems) [8, 9], are used both in the products sold by OSD and within PARC. Each protocol world includes a hierarchy of protocols for various purposes such as transporting files, or sending and receiving mail.

To further complicate things, there are two Internets within Xerox: old and new. The new one, the Corporate Internet, is run by the branch of Xerox that provides things like our phone service. It's been growing quite rapidly over the past year. The idea is to use the official products from SDD to support the non-research people within Xerox and to demonstrate how well our equipment really works. The products don't support PUPs, so neither does the Corporate Internet.

The Research Internet, the "old" Internet developed by the research community, is the collection of ethernets, phone lines, and gateways that carry PUPs. (The Research Internet hardware also transports a lot of NS packets just to keep you on your toes.) There are phone lines as far as Japan and England, with major clusters of workstations in El Segundo (think LA) and Rochester (NY). Within the Palo Alto area, 56KB and T1 lines connect various buildings. (T1 is a

1.5 megabit/sec phone line. That's faster than the software in the gateways. Think of it as a very long skinny Ethernet.)

In addition to connecting up all of the personal computers, the network also includes a number of machines generically called *servers.* Normally, servers have special purpose, expensive hardware attached to them, such as large-capacity disks, or printers. Their purpose in life is to make that hardware available to the local community. We tend to identify servers by function, so we talk about print servers, file servers, name lookup servers, mailbox servers, tape servers, and so on. Many of the protocols for use of the Ethernet were developed precisely so that personal computers could communicate effectively with servers.

The Alto

The innards of the Alto are wonderfully described in a clear and informative blue-and-white report [10]; I seriously recommend that you read it. In the very unlikely event that you need to know still more about the Alto, you might try looking in the Alto hardware manual <11>. But for our purposes, suffice it to say that the Alto is a 16-bit minicomputer whose primary claim to fame is that it comes equipped with a bitmap display, a mouse, and an Ethernet interface.

D-Machines

The D-machines are a family of personal computers, each member of which has a name starting with the letter "D". As long as you don't look too closely, D-machines look a lot alike. In particular, they are all 16-bit computers with a microprogrammed processor that handles most of the I/O as well running the user's programs. And they all generally come equipped with a hard disk, a bitmap display, a keyboard, a mouse, and an Ethernet interface. There are differences of course: in size, in speed, and in flexibility.

The Dolphin (formerly called the D0)

The Dolphin was one of the early D-machines, and there are still a few of them around. Dolphins are housed in the same sized chassis as Altos. You can tell that they aren't Altos because they have wide screen terminals, and because they don't have a slot on top for a removable disk pack. Instead, they use a 28MByte Winchester disk drive made by Shugart. Dolphins can talk to both 3 MBit and 10 MBit Ethernets.

The Dandelion

The Dandelion is the D-machine processor that is used in the Star products. It comes in a box about half the width of an Alto chassis, and roughly the same height and depth. Dandelions are less flexible than Dolphins, since the microprocessor is shared among the various I/O devices and the emulator in a fairly rigid round-robin fashion (associated with the terms "clicks" and "rounds"). As a consequence, it isn't very easy to hang a new I/O device off of a Dandelion. On the other hand, Dandelions are both faster and cheaper than Dolphins. Dandelions talk only to 10 MBit Ethernets.

The Dorado

Building large software systems is a demanding chore. It doesn't help any when the hardware upon which your programming environment is based doesn't have enough horsepower to support you properly—that is, in the manner to which you would like to become accustomed. After some

years of trying to shoehorn large programs into Altos, CSL twisted the arms of its hardware folk and talked them into building the Dorado, the current high-performance model in the D-machine line. The processor, the instruction fetch unit, and the memory system of the Dorado have been written up in papers for your enjoyment [12]. Dorados come equipped with an 80 MByte removable disk pack or a 315 MByte Winchester drive, and talk only to 3 MBit Ethernets at present.

A Dorado is roughly three to five times faster than an Alto when emulating an Alto, that is, running BCPL. A Dorado runs compute-bound Mesa software roughly eight to ten times as fast as an Alto. Because of the raw power of a Dorado, it is usually the computer of choice for substantial programming projects. The primary difficulty about Dorados is that there aren't enough of them (and the related fact that they are rather tricky to build). Many people have their own, but others must share a pool of public machines. Now, even though most Dorados have removable packs, it really isn't very convenient to start your session of a public Dorado by mounting your own pack. The biggest difficulty is that you must be at the processor to change the disk pack, and the processor is a long way away. Subsidiary difficulties are that you must power down a Dorado in order to change the disk pack, and that T-80 disk packs are difficult to label effectively. As a result, when you borrow a Dorado, you generally also want to borrow at least some of the space on that Dorado's local disk. In order for this sharing to work out well, certain social taboos and customs concerning the use of such local disks have emerged, under the general rubric of "living cleanly". More on this topic anon.

In a return to the ways of the past, the Dorado processors are rack mounted in a remote, heavily air-conditioned machine room. It was initially intended that the Dorado, like the Alto, would live in your office. To prevent its noise output from driving you crazy, a very massive case was designed, complete with many pounds of sound-deadening material. But experience indicated that Dorados ran too hot when inside of these cabinets, and the concept of having Dorado processors in offices was abandoned. With progress in general and VLSI in particular, there is hope that the successor to the Dorado (Dragon) will once again come out of the machine room and into your office.

The Dicentra

The Dicentra is the newest D-machine. Essentially, it consists of the processor of the Dandelion with the tasking stuff striped out squeezed onto one Multibus card. It communicates with its memory and with I/O devices over the Multibus. Dicentras will talk to any Ethernet, or any I/O device for that matter, for which you can supply a Multibus interface card; that's one of the Dicentra's strengths. The initial application of the Dicentra is as a processor for low cost Internet gateways. The Dicentra and the Dandelion are named after wildflowers partially because they are outgrowths of an initial design of Butler Lampson's called the Wildflower.

The Dragon

The Dragon is a high-performance multi-processor machine being designed and implemented by the design group of CSL using CMOS integrated circuit technology. Confusingly enough, the Dragon is *not* a D-machine: it is a 32-bit machine, embodying RISC and snoopy-cache design principles. Compilers for Dragon will produce Dragon machine code rather than Cedar/Mesa byte codes. While there is microcode in the Dragon, it is built into the processor and is not environment-specific in the way that D-machine microcode is.

A Dragon processor executed the first Dragon instructions in early 1987. A "softcard" allows a single Dragon processor to sit in a 6085 and make use of its peripherals during Dragon hardware

and software debugging. The first multi-processor Dragon workstation is called the June87 Machine (expected late in 1987). It will have a fast bus, custom caches interfacing the processors to the bus, and a custom display controller.

The process of porting Cedar to the Dragon has begun. The Mimosa compiler produces Dragon machine code from Cedar language programs, and the implementation of the Cedar runtime environment is well under way.The Dragon is a high-performance multi-processor machine being custom designed using CMOS integrated circuit technology by the design group of CSL (formerly the Integrated Design Laboratory). Confusingly enough, the Dragon is not really a D-machine. For example, the Dragon word size is 32 bits rather than 16. The underpinnings of Cedar will be adjusted as necessary so that Cedar will run on a Dragon; but this will take some doing.

A few comments about Booting

All of the local processors come equipped with a hidden button called the "boot button" that is used to reinitialize the processor's state. The Alto had just one boot button, hidden behind the keyboard; pushing it booted the Alto. On Dolphins, the situation is only slightly more complex: there are two boot buttons, one at the back of the keyboard, and the other on the processor chassis itself. They perform roughly the same function, but the one on the chassis is a little more potent. On Dorados, there is a lot more going on. There are really two computers involved, the main Dorado processor and a separate microcomputer called the *baseboard.* It is the baseboard computer's job to monitor the power supplies and temperature and to stage-manage the complex process of powering up and down the main processor, including the correct initialization of all of its RAM's. The boot button on a Dorado is actually a way of communicating with this baseboard computer. You encode your request to the baseboard computer by pushing the boot button repeatedly: each number of pushes means something different. For details, see Ed Taft's memo on the subject <13>. If the baseboard computer of the Dorado has gone west for some reason (as occasionally happens), your only hope is to push the *real* boot button, a little white button located on the processor chassis itself, far, far away. Just as the boot button on the keyboard is essentially a one-bit input device for the baseboard computer, the baseboard computer also has a one-bit output device: a green light located on the processor chassis. Various patterns of flashing of this light mean various things, as detailed in <13>.

There is one more bit of folklore about booting that I can't resist mentioning—every once in a while, I have to throw in some subtle tidbit to keep the wizards who read this from getting bored. Our subject this time is the "long push boot". Suppose that you have been working on your Dorado for a while, and you walk away to go to the bathroom. When you return and reach toward your keyboard, you get a static shock. You are only mildly annoyed at this until you notice that the cursor is no longer tracking the mouse, and the machine doesn't seem to hear any of your keystrokes. The screen looks OK, but the Dorado is ignoring all input. What has probably happened is that the microprocessor in your terminal has been knocked out by the static shock. Yes, Virginia! In addition to the Dorado itself, and the baseboard computer, there is also a microprocessor in your terminal (located in the display housing), which observes your input actions and sends them on to the main processor under a protocol referred to as "the seven-wire interface". What you want to do now is to reboot the terminal microprocessor without disturbing the state of the Dorado at all—after all, you were in the process of editing something, and you are now in danger of losing those edits. What you should do is to depress the boot button and hold it down for quite a while (more than 2.5 seconds): and then release it. This is known as a "long push boot", and it does just what you want under these conditions: it reboots your terminal without affecting anything higher up.

Local Programming Environments

Various programming environments have grown up around the various pieces of hardware mentioned above. You can get a software merit badge simply by writing one non-trivial program in each environment.

BCPL

The first high-level programming language used on the Alto was BCPL, and quite a bit of program writing was done in that environment over the years. By now, however, essentially no new programming is being done in BCPL. The language itself will be around for some time to come, since there are BCPL programs that perform valuable services for us: the print server programs Press and Spruce and the file server program IFS are three important examples.

History: Of the better-known computer languages, BCPL is closest to C. The fundamental data type in BCPL is a sixteen-bit word. There are facilities in the language for building structured data objects including records and pointers. But there is no type-checking in the language at all. For example, if *foo* is a pointer to a record of type *node* that includes a field named *next*, that field is referenced in BCPL by writing

"foo>>node.next",

which means "treat *foo* as a pointer to a *node*, and extract the *next* field". In a strongly typed language, you wouldn't have to mention that *foo* was a pointer to a *node*, since the compiler would be keeping track of the fact that *foo* was so declared. The BCPL compiler, however, thinks of *foo* as a sixteen bit value, just like any other sixteen bit value. For example, it would be legal in BCPL to write

"(foo+7)>>node.next", or "foo>>otherNode.next".

Some of the strictness of the Mesa approach to type-checking and version matching discussed below may be a reaction to BCPL's free-wheeling ways of handling these issues. Further details about the BCPL language and environment can be found elsewhere <15, 16, 17>.

The debugger in the BCPL environment was named "Swat". This name is preserved in the local dialect as the name of the bottom of the three unmarked keys at the right edge of the keyboard. Various debuggers may be invoked in various environments by depressing this key, perhaps in conjunction with the left-hand shift key or the control key. (The right hand shift key won't do: it is too close to the swat key itself for comfort!)

Mesa

Mesa is a strongly typed, PASCAL-like implementation language designed and built locally. It first ran on Altos. Herein, I shall call that system Alto/Mesa. Dolphins and Dorados (but not Dandelions) can run Alto/Mesa by impersonating an Alto at some level. More recent instances of Mesa now run on all of our D-machines under the Pilot operating system. In passing, I should observe that Pilot is an operating system written in Mesa by folk in SDD. It is a heavier-weight operating system than the Alto OS, providing its clients with multiprocessing, virtual memory, and mapped files.

History: Alto/Mesa programs do not use the Alto OS at all, mostly because Mesa and BCPL have rather different philosophies about the run-time world in which they exist. So the first thing that a Mesa program does when running on an Alto is to junta away almost all of the OS, and set about building a separate Mesa world. It is a considerable nuisance for Mesa and BCPL programs to communicate, since their underlying instruction sets are completely different. So, most of the important OS facilities, such as the file system, had to be re-implemented directly in Mesa. Mesa's memory management strategies replace the revolutionary tactics of "junta" and "counter-junta" with the relative anarchy of segment swapping.

A fair amount of software was written in Alto/Mesa, but little new programming is being done in that environment; that is, Alto/Mesa isn't quite as dead as BCPL, but it is getting there. Perhaps the crown jewels of Alto/Mesa are the systems Laurel, Grapevine, Mockingbird, and Griffin. You will be hearing more about the former two in the section on electronic mail; to find out more about the latter two, check out their entries in the Glossary.

The Pilot version of Mesa is the home to lots of active programming in several locations. First, it is the system in which the Star product was and is being implemented by OSD. The programmers in OSD have developed a set of tools for programming in Mesa variously called the "Tools Environment" or "Tajo". This body of software may soon be marketed under the name "the Xerox Development Environment" and is currently being released to numerous Universities in the most recent incarnation of the University grant program.

Although Mesa programs look a lot like PASCAL programs when viewed in the small, Mesa provides and enforces a modularization concept that allows large programs to be built up out of smaller pieces. These smaller pieces are compiled separately, and yet the strong type checking of Mesa is enforced even between different modules. The basic idea is to structure a system by determining certain abstract collections of facilities that some portions of the system will supply to other portions. Such an abstraction is called an "interface", and it is codified for the compiler's benefit in a Mesa source file called an "interface module". An interface module defines certain types, and specifies a collection of procedures that act on values of those types. Only the procedure headers go into the interface module, not the procedure bodies (except for INLINE's, sad to say). This makes sense, since all the interface module has to do is to give the compiler enough information so that it can type-check programs that use the abstraction.

Having specified the interface, some lucky hacker then has the job of implementing it—that is, of writing the procedure bodies that actually do the work. These procedure bodies go into a different type of module called an "implementation module". An implementation module is said to "export" the interface that it is implementing; it may also "import" other interfaces that it needs to do its job, interfaces that some other program will implement.

In simple systems, each interface is exported by exactly one module. In such a system, there isn't much question about who should be supplying which services to whom. In fact, in these simple cases, the *binding*, that is, the resolution of imports and exports, can be done on the fly by the loader. But in more complex cases, there might be several different modules in the system that can supply the same service under somewhat different conditions, or with somewhat different performance. Then, the job of describing exactly which modules are to supply which services to which other modules can become rather subtle. A whole language was devised to describe these subtle cases, called C/Mesa. The Binder is the program that reads a C/Mesa description, called a *config*, and builds a runnable system by filling imports request from exports according to the recipe.

The Mesa language is described in the recently updated manual published by the Systems Development Department of OSD [18]. Older blue-and-white versions still abound, and may be of use if the newer version is not available [19]. You may also be interested in Jim Morris's comments on how programs should be structured in Mesa [20].

Smalltalk

Smalltalk was developed by the folk in the System Concepts Laboratory (formerly known as the Software Concepts Group). The Smalltalk language is the purest local embodiment of "object-oriented" programming:

A computing world is composed of "objects".

The only way to manipulate an object is to be polite, and ask it to manipulate itself. One asks by sending the object a message. All computing gets done by objects sending messages to other objects.

Every object is an "instance" of some "class".

The class definition specifies the behavior of all of its instances—that is, it specifies their behavior in response to the receipt of various messages.

Genealogists will recognize that ideas from both Simula and Lisp made their way into Smalltalk, together with traces of many other languages.

For some years now, the folk in SCL have been working at trying to get the Smalltalk language and system out into the great wide world. The first public event that came out of this effort was the August 1981 issue of Byte magazine; it was devoted to Smalltalk-80, including a colorful cover drawing of the now famous Smalltalk balloon. In addition, the SCL folk have written several books about Smalltalk, and they are planning to license the system itself to various outside vendors. The first of the books, entitled *Smalltalk-80: The Language and Its Implementation*, emerged from the presses at Addison-Wesley in 1983 [21]. Other books include *Smalltalk-80: The Interactive Programming Environment*, and *Smalltalk-80: Bits of History, Words of Advice.*

Interlisp-D

LISP is the standard language of the Artificial Intelligence community. Pure LISP is basically a computational incarnation of the lambda calculus; but the LISP dialects in common use are richer and bigger languages than pure LISP. Interlisp is one dialect of LISP, an outgrowth of an earlier language called BBN-LISP; for more historical details, read the first few pages of the Interlisp Reference Manual [22]. One of the biggest strengths of Interlisp is the large body of software that has developed to assist people programming in Interlisp. Consider the many features of Interlisp: an interpreter, a compatible compiler, sophisticated debugging facilities, a structure-based editor, a DWIM (Do What I Mean) error correction facility, a programmer's assistant, the CLISP package for Algol-like syntax, the Masterscope static program analysis database, and the Transor LISP-to-LISP translator, to name a few.

Interlisp itself has been implemented several times. Interlisp-10 is the widely-used version that runs on PDP-10's. Interlisp-D is an implementation of Interlisp on the D-machines [23], produced by folk at PARC. In the process of building Interlisp-D, the boundary between Interlisp and the underlying virtual machine was moved downward somewhat, to minimize the dependencies of Interlisp on its software environment; that is, functions that were considered primitive in Interlisp-10 were implemented in Lisp itself in Interlisp-D. But the principal innovations of Interlisp-D are the extensions that give the Interlisp user access to the personal machine computing environment: network facilities and high-level graphics facilities (including a window package) among them.

By the way, Interlisp has the honor of being the first system (to my knowledge) to use the prefix "Inter-". This prefix has become quite the rage of late: Internet, Interpress, Interscript—you get the general idea.

Cedar

Back in 1978, folk in CSL began to consider the question of what programming environment we would use on the emerging D-machines. A working group was formed to consider the programming environments that then existed (Lisp, Mesa, and Smalltalk) and to form a catalog of programming environment capabilities, ranked by both by value and by cost. A somewhat

cleaned-up version of the report of that working group is available as a blue-and-white for your perusal [24]. After pondering the alternatives for a while, CSL chose to build a new programming environment, based on the Mesa language, that would be the basis for most of our programming during the next few years. That new environment is named "Cedar". We refer the interested reader to [40] for an introductory "tour" through the Cedar programming environment and to [42] for an in-depth description of the overall structure of Cedar and the way it is organized.

Cedar documentation is in a constant state of flux; indeed, it might be said that Cedar as a whole, not only its documentation, is in a constant state of flux. Much of the documentation for the current release is accessible through a ".df" file named Documentation.df <25>. Hardcopies of this packet of stuff, entitled "The Cedar Manual", are produced from time to time, and distributed to Cedar programmers.

The programming language underlying Cedar is essentially Mesa with garbage collection added. Now, adding garbage collection actually changes things quite a bit. First of all, it changes programming style in large systems tremendously. Without garbage collection, you have to enforce some set of conventions about who owns the storage. When I call you and pass you a string argument, we must agree whether I am just letting you look at my string, or I am actually turning over ownership of the string to you. If we don't see eye to eye on this point, either we will end up both owning the string (and you will aggravate me by changing *my* string!) or else neither of us will own it (and its storage will never be reclaimed—a storage leak). Once garbage collection is available, most of these problems go away: God, in the person of the garbage collector, owns all of the storage; it gets reclaimed when it is no longer needed, and not before. But there is a price to be paid for this convenience. The garbage collector takes time to do its work. In addition, all programmers must follow certain rules about using pointers so as not to confuse the garbage collector about what is garbage and what is not.

Thus, programs in the programming language underlying Cedar look a lot like Mesa programs, but they aren't really Mesa programs at all, on a deeper level. To avoid confusion, we decided to use the name "Cedar" to describe the Cedar programming language, as well as the environment built on top of it. Cedar is really two programming languages: a restricted subset called the *safe language*, and the unrestricted full language. Programmers who stick to the safe language can rest secure in the confidence that nothing that they can write could possibly confuse the garbage collector. Their bugs will not risk bringing down the entire environment around them in a rubble of bits. Those who choose to veer outside of the safe language had better know what they are doing.

Those who want to know more about Cedar are once again encouraged to dredge up a copy of the Cedar Manual <25>. It includes documentation on how Cedar differs from Mesa, annotated examples of Cedar programs, manuals for many of Cedar's component parts, a Cedar catalog, and lots of other good stuff. By the way, the most authoritative source for what the current Cedar compiler will do on funny inputs can be found in a document called the Cedar Language Reference Manual, also known by the acronym CLRM. This is logically part of the Cedar Manual, but it is currently bound separately, and only available in draft form. The CLRM suggests a particular design philosophy for building a polymorphic language that is a superset of the current Cedar, since that is the direction in which the authors of the CLRM, Butler Lampson and Ed Satterthwaite, wanted to nudge the Cedar language.

Local Software

This section is a once-over-lightly introduction to some of the major software systems that are available in the Alto and Cedar worlds. First, let me mumble some general words about how such subsystems are documented. The most commonly used Alto subsystems are documented in a tome called the Alto User's Handbook [26]. The less commonly used ones are documented in a catalog entitled "Alto Subsystems" <27>. In addition, Suzan Jerome wrote a Bravo primer aimed at non-programmers [28]. In Cedar, the current best sources are the Cedar Manual mentioned above <25>, and a public database, sitting on Alpine, containing whiteboards of Cedar documentation. The whiteboards system provides graphical arrangement and access of information.

Filing

When programming in the Alto world, or in current Cedar, you are dealing with two different types of file systems: local and remote. The local file system sits on your machine's hard disk. Remote file systems are located on file servers, machines with big disks that are willing to store files for you. Local file systems have several unpleasant characteristics in comparison with the remote systems: they are small, and they aren't very reliable. Both of these problems have consequences.

Most systems, including Cedar, keep at least one or more old versions. Remote file servers also maintain multiple versions of files, and letting old versions of things accumulate is one easy way to overflow your disk usage allocation.

No disk is completely reliable. Our remote file servers have automatic backup facilities that protect us from catastrophic disk failures. But the local file systems have no such automatic protection. Since this protection isn't provided automatically, it behooves you to adjust your behavior appropriately: make sure that, on a regular basis, backup copies of the information on your local disk are put in some safe place, such as on a remote file server where suitable precautions are constantly being taken by wizards to protect against disk failure. Doing this is one facet of what is meant by the phrase *Living Cleanly*, which deserves its own section.

Living Cleanly (also known as "Keeping your bags packed")

The phrases "living cleanly" and "keeping your bags packed" refer to a particular style of use of your local file system. In order to understand the cosmic issues involved, we should pause to discuss the ways in which local and remote file systems have been used over the years.

History: Back in the Alto days, personal files were usually stored on one's Alto disk pack, while project-related and other public files were stored on remote servers. Careful folk would occasionally store backup copies of their personal files on remote servers as well, in case of a head crash. But, as a general rule, one thought of one's Alto pack as the repository of one's electronic state. This made sharing Altos quite convenient, since you could turn any physical Alto into "your Alto" just by spinning up your disk pack.

In the glorious world of the Cedar future, all of your personal files as well as all public files will live on file servers in the network. The disk attached to your personal computer will, from time to time, contain copies of some of this network information, for performance reasons; but you won't have to do anything to achieve this, and you won't have to worry about how it is done. From the user's point of view, all files will act as if they were remote at all times. Indeed, except in a few funny cases, there won't even be any notion of "local file"; "file" will mean "remote file".

At the moment, we are sitting in an unpleasant transitional phase somewhere between these two styles of usage of the local disk: we are attempting to simulate the latter state by means of manual methods and social pressure. We want you to think of your data as really living out on the file servers. That is the proper permanent home for your personal files as well as for public files. You will have to bring copies of these files, both private and public, to your local disk in order to

work on them. But, at the end of each editing session, you should store the new versions of files that you have created back out to their permanent remote homes. None of this happens automatically at present; you have to make it happen manually by using "DF files" discussed below. Learning how to use these tools can be confusing, but there are four important benefits to be reaped from adopting a clean living life-style.

First, you are taking a step towards the glorious future.

Secondly, you are protecting yourself against failures of the local disk. A clean liver only holds information on her local disk for the duration of an editing session. This puts a reasonable bound on the amount of information that she can lose because of a disk crash.

Thirdly, there are various reasons why erasing your local disk is a good idea when updating to a new release of the Cedar system; sometimes, in fact, it is required. Since clean living folk don't keep long term state on their local disks, this doesn't bother them in the slightest.

Finally, and perhaps most importantly, clean living is the key to sharing disk space on machines without removable disks. When you use a public Dorado or Dandelion, you are forced to share its disk space with the other members of the community. This sharing is predicated on a policy of clean living: when your session is over, you must store away all of your files on remote file servers. The person who uses the machine next may need to free up some disk space; if so, she is perfectly entitled to delete your files without qualm or pause. And you won't mind a bit, it says here, because you have been living cleanly.

The above paragraph is the "letter of the law" regarding the sharing of public disk space, also known as the "true religion". A document exists for followers and believers of the true religion <39>. People who want to be well regarded should pay some attention to this doctrine: sharing things is always more pleasant when everyone acts with a modicum of politeness and care. Don't delete the previous user's files if she was called away by some disaster and didn't have a chance to clean up. Try not to delete the standard system files, such as the Compiler, that sit in the local file system, since whoever follows you will be justifiably aggravated by their absence. Even more important, if you do exotic things such as bringing over non-standard versions of system files, try to put everything back to normal when you leave ere you cause whoever follows you to become hopelessly confused.

Local file systems

History: The local file system in the Alto world is called either the "Alto file system" or the "BFS", the latter being an acronym for Basic File System. The biggest that a BFS can be is 22,736 pages. This is substantially bigger than the entire disk on an Alto. However, Dolphins and Dorados have much bigger local disks. Hence, when a Dolphin or Dorado is emulating an Alto, its local disk is split up into separate worlds called *partitions*, each containing a maximum-sized BFS. Dolphin disks can hold two full partitions, while Dorado disks can hold either five or nineteen (Dorado disks store either 80 or 300 MBytes). What partition you are currently accessing is determined by the contents of some registers that the disk microcode uses. There is a command called "partition" in the Alto Executive and the NetExec that allows you to change the current partition.

When operating in the Cedar world, a disk pack is called a physical volume, and it is divided into worlds called logical volumes. The area of the disk devoted to Cedar volumes must be disjoint from the area devoted to Alto-style partitions. Some Dorados that run Cedar have one full-sized Alto partition, and the rest of the disk devoted to Cedar; there are also many Dorados that have the entire disk devoted to Cedar.

You may hear the terms "labeled file system" or "labeless file system" bandied about in the context of file systems and disk discussions. These terms describe the two file representations

supported by our local file systems.

The so-called labeled file system relies on special disk hardware; this hardware stores a software "label" on every disk sector. The basic idea is that you must tell the disk not only the address of the sector you want to read or write, but also what you think that sector holds. This makes it difficult, though not impossible, to write in the wrong place.

There are two unfortunate drawbacks of the labeled file system. Most important, it relies on specialized disk hardware. Nowadays, it is *almost* impossible to buy disks with labels for any reasonable price. In addition, the labeled file system performs rather poorly as label maintenance is time consuming. Understandably then, the Dragon will not support labeled disks and in preparation for this change, a labeless file system [43] was developed.

The labeless file system has nearly the same semantics as its labeled predecessor. One slight difference is that updates to the meta-data of the file system are logged and committed about every half second. That is, file creates, deletes, extends, contracts, cache loads, cache flushes, update to the last used time for cache files, and so on do not *permanently* happen until the changes are logged – about twice a second. As long as the system continues to run, the effect of this is nil – except performance is much better. In the event of a crash, no more than the last 1/2 second of file changes may be lost.

Cedar treats its local disk space in a novel way, which you can learn about either now, or through painful experience. If a file is created on the local disk, it is a normal file and behaves in the expected ways. If, however, the file came from a file server, or is stored onto a file server, it becomes an *attachment*. For most purposes, such a file acts like a normal file on your disk, but since the file system ardently believes that the version on the file server is the truth, it is free to delete the local copy of the bits from your local disk wherever it needs the space for something else. If you later try to open the file again, it will fetch the file from the server. This whole scheme works well, **unless you delete files from the server!** If you do, the assumptions of the Cedar file system go out the window, and you can lose files. Of course, you will need to delete files sometime, to avoid running out of space, but for the most part you will want to delete only the older versions of your files. (There is a command called KillExcessVersions to help you with this chore).

If you have skipped to the fine print, go back and read the above paragraph.

Remote file systems

The most important local file servers are IFS's, an acronym for Interim File System (one of the crown jewels of the BCPL programming environment). Like I always say, "temporary" means "until it breaks", and "permanent" means "until we change our minds". Cyan, Indigo and Ivy are three prominent local IFS's: Cyan and Indigo store mostly project files, while Ivy stores mostly personal files. Juniper was CSL's first attempt to build a distributed transactional file server: it was one of the first large programs written in Mesa. Alpine is a new effort to build such a beast in the context of Cedar, in support of distributed databases and other such wonderful things [41]. Most Walnut users store their mail databases on Alpine.

There is no coherent logic to the placement of "general interest" files and directories, nor even to the division between Cyan, Indigo and Ivy. Browse through the glossary at the end of this document to get a rough idea of what's around. IFS supplies a general sub-directory structure, and as a result there are lots of places to look for a file on an IFS. For example, on Indigo you might look for

[Indigo]<Packages>Doc>MyFavoritePackage.press, or

[Indigo]<Packages>MyFavoritePackage>Documentation.press, or

[Cyan]<Cedar6.1>Documentation>MyFavoritePackageDoc.tioga, or

[Cyan]<CedarChest6.1>Documentation>MySecondFavoritePackageDoc.tioga

or perhaps some other permutation. This requires a bit of creativity and a little practice. However, if you get in the habit of using "*"s in file name specifications, you will find all sorts of things you might not otherwise locate. Note that a "*" in a request to an IFS will expand into all possible sequences of characters, *including* right angle brackets and periods. Thus, for example, a request for

<Packages>*press

refers to all files on all subdirectories of the Packages directory that end with the characters "press". A "*" won't match a left angle bracket, by the way. Thus, if you ask for "*.press", you are referring to all Press files on the current directory. If you ask for "<*.press", you are referring to all of the Press files on the entire IFS (expect such a search to take a long time!).

For Cedar, the first places to look for documentation are on

[Cedar]<Cedar6.1>Documentation>*Doc.tioga, and

[Cedar]<CedarChest6.1>Documentation>*Doc.tioga,

where the * is usually just the package name. For a compilation of abstracts, commands, and keywords, consult

[Cedar]<Cedar6.1>Documentation>CedarCatalog.tioga

[Cedar]<CedarChest6.1>Documentation>CedarChestCatalog.tioga

Within Cedar, the file system supports an alternative UNIX-like file naming convention, which replaces the various flavors of brackets with slashes. Thus the documentation file mentioned above could be referred to as

/Indigo/Packages/MyFavoritePackage/Documentation.press.

Pseudo-servers

The Cedar system supports the notion of "Pseudo-servers" in file names: the most common ones you will see are 'Cedar', 'Fonts', and 'User'. These are not the name of any actual file server (we hope!), but are used to refer to whichever locally-available server is fullfilling the corresponding function. For instance, you would normally refer to this document by the name

[Cedar]<CedarChest6.1>Documentation>BriefingBlurb.tioga

and let the system translate the name to the actual server (usually Cyan in this case). In the case of directories whose contents don't change very often, this facility is also used to provide automatic (or rapid manual) switchover to a secondary server, in case the primary server is down.

File Properties

The "size" of a file is its length measured in disk pages: the "length" of a file is its length

measured in bytes. The "create date" of a file is the date and time at which the information in that particular version of the file was "created", that is, the date when this that sequence of bytes came into being. Copying a file from one file system to another does not change the create date, since the information in the file, the sequence of bytes, is not affected. The create date is almost always what you want to know about a file. Some of our systems also maintain a "write date" or a "read date", but they are less well defined, and not as interesting.

Editing and Typesetting

In the outside world, document production systems are usually de-coupled from text editors. One normally takes the text that one wants to include in a document, wraps it in mysterious commands understood by a document processor, feeds it to that processor, and puzzles over the resulting jumble of characters on the page. In short, one programs in the document processor's language using conventional programming tools—an editor, a compiler, and sometimes even a debugger. Programmers tend to think this is neat; after all, one can do anything with a sufficiently powerful programming language. (Remember, Turing machines supply a sufficiently powerful programming language too.) However, document processors of this sort frequently define bizarre and semantically complex languages, and one soon discovers that all of the time goes into the edit/compile/debug cycle, not careful prose composition.

History: Bravo is the editor and typesetter in the Alto world, and it represented a modest step away from the programming paradigm for document production. A single program provided both the usual editing functions *and* a reasonable collection of formatting tools. You can't program Bravo as you would a document "compiler", but you can get very tolerable results in far less time. The secret is in the philosophy: what you see on the screen is what you get on paper. You use the editing and formatting commands to produce on the screen the page layout you want. Then, you tell Bravo to ship it to a print server and presto! You have a hardcopy version of what you saw on the screen. Sounds simple, right?

Of course, it isn't quite that easy in practice. There are dozens of subtle points having to do with fonts, margins, tabs, headings, and on and on. Bravo was a success because most of these issues are resolved more or less by fiat—someone prepared a collection of configuration parameters and a set of forms that accommodated most document production. Many of the configuration options aren't even documented, so it is hard to get enough rope to hang yourself. The net effect is that one spent more time composing and less time compiling.

In Bravo's wake, several new editors of unformatted text appeared: the Laurel editor, and the editor in the Tools Environment are prominent examples. The Laurel editor is particularly noteworthy in that it pioneered the development of a modeless (or at least less modal) user interface for an editor. The Star product editor and Tioga are more recent local editors in the full Bravo tradition: they can handle formatting and multiple fonts. Tioga is the editor within Cedar, and its user interface is very close to the widely beloved Laurel modeless interface—try going back to Bravo after using Tioga for a while, and see how horrible it feels to have to remember to type "i" and "ESC" all the time. Tioga shows formatted text on the screen. To get a hardcopy of that text, the current path involves running a companion program called the XTSetter [50], which will compose your pages for printing and send them to a print server. We expect the method of getting things printed to change shortly: consult the online documentation for the latest scoop.

Dealing with editor bugs

All text editors have bugs. Furthermore, you are often most likely to tickle one of the remaining bugs in an editor when you are working furiously on a hard problem, and hence, have been editing for a long time without saving the intermediate results. As fate would have it, these are exactly the times when it is most damaging and most upsetting to lose your work. There is nothing quite like the sinking feeling you get when a large number of your precious keystrokes

gurgle away down the drain. Both Bravo and Tioga have mechanisms that can, in some cases, save you from the horrible fate of having to do all those hours of editing over again.

History: Bravo attempts to safeguard you by keeping track of everything that you have done during the editing session in a log file; in case of disaster, this log can be replayed to recapture most of the effects of the session. If you have a disaster when editing in Bravo, be careful NOT to respond by running Bravo again to assess the damage. By running Bravo again in the normal way, you will instantly sacrifice all chance of benefiting from the log mechanism, since the log allows replay only of the most recent session. What you want to do instead is run the program "BravoBug" ("Bravo/R" is not an adequate substitute). It wouldn't be a bad idea to ask a wizard for help also. While you are looking for a wizard, try and think of some good answer to the question "Why are you using Bravo, anyway?", which said wizard will almost certainly ask.

The most common—perhaps I should really say "the least rare"—source of editing disasters in Tioga is problems with monitor locks. Unfortunately, this class of problem usually makes further progress in any part of Cedar impossible, since Tioga is so basic to the Cedar system. If you get into this situation, you can often recover by holding down both the left and the right shift keys and the Swat key for more than 1 second; this will call ViewerOpsImpl.SaveAllEdits[] to save the contents of all the viewers. While the saving is taking place, the cursor will become a black box. You may then re-boot and pick up (almost) where you left off. Unnamed viewers get saved, too; consult TiogaDoc to find out where. If you are reading this online, you might want to make a note of this now, because you won't be able to open this document when you need to.

Non-Textual Editors

We humans often convey information more effectively in ways other than the written word. Cedar likewise provides tools for manipulating representations of information other than text, such as 2-D illustrations, mathematical expressions and voice.

Gargoyle[45] is an interactive 2D illustrator for creating color pictures. It includes novel features to aid the user in precise geometric placement of objects in the scene. These features are called "snap-dragging" and "alignment objects." Refer to [46] for an on-line tutorial.

CaminoReal[44] is an environment for two kinds of manipulations of mathematical expressions: (1) interactive, syntax-directed, two-dimensional, WYSIWYG editing and (2) algebraic manipulation.

As part of the Cedar Voice project[47], a tool for voice editing was developed that uses a record/stop/backup model. Editing is done largely at the phrase level (never at the phoneme level), representing the granularity at which editing can be done fastest and with least effort.

Tioga provides a rich enough common ground so that users can integrate heterogenous data (e.g. voice, pictures, equations) into a single source (namely a Tioga document) for later browsing. Besides being able to display text, Tioga has enough smarts to display bitmaps or even to speak voice. The other editors, of course, recognize Tioga's role as chief editor and each provides a "stuff" facility so that users can deposit locally editable information into a Tioga document.

Printing

In general, our printers are built by taking a Xerox copier and adding electronics and a scanning laser that produce a light image to be copied. There are many different types of such printers, and there are multiple instances of each printer type as well. There are also many different programs that would like to produce printed output. The Press print file format was our

first answer to the problem of allowing every printing client to use every printer. The Interpress [51] format is a newer corporate standard that fulfills much the same purpose. Interpress and Press files are the Esperanto of printing. Most print servers demand that the documents that you send to them be in one of these formats. This means you have to convert whatever you have in hand (often text) to the appropriate format before a server will deign to print it.

Press file format <30> is hairy, and some print servers don't support the full generality of Press. Generally, however, such servers will simply ignore what they can't figure out, so you can safely send them any Press file you have.

A Press file can ask that text be printed in one of an extensive collection of standard fonts. Unfortunately, you must become a wizard in order to print with your own new font. You can't use a new font unless it is added to the font dictionary on your printer, and adding fonts to dictionaries is a delicate operation: a sad state of affairs. If the Press file that you send to a printer asks for a font that the printer doesn't have, it will attempt a reasonable substitution, and, in the case of Spruce, tell you about the substitution on the break page of your listing. If you have chronic font difficulties of this sort, contact a wizard.

The newer print file format is Interpress. The print servers that are part of the Star product speak a dialect of Interpress. A print file in Interpress format is called a *master*.

One route for printing Interpress masters involves converting them first into a printer-dependent print file in so-called PD format (with conventional extension ".pd"). From there, a relatively simple driver program on each printer produces the final output.

PARC has a variety of printers available for your hardcopy needs. We have high volume printers for quantities of text, listings, and documentation; we have slower printers with generally higher quality for more complex files; and we have very slow printers for extremely high quality.

All of our current press printers offer 384 spots per inch and share a common font dictionary. We use two different software systems for printing Press files, both running on Altos: one is called Spruce, and the other is called (confusingly) Press. Spruce offers speed and spooling, but it can only image characters and rules, and not too many of them. This makes it limited in graphics applications. Furthermore, Spruce is limited to the particular sizes of fonts that it has stored in its font dictionary: it does not know how to build new sizes by converting from splines. Press is slower, but can handle arbitrary bitmaps, and can produce odd-sized fonts from splines.

The imaging group has developed Interpress printing capabilities. Printing ".pd" files is now an option on most Press printers (that is, on printers running the program Press as opposed to Spruce). Just ship your ".pd" file to the printer in the standard way: it is smart enough to figure out whether what you have sent it is in PD or Press format, and it will invoke PDPrint or Press as appropriate. Documentation on these two printing programs is available, by the way <31, 32>. There are newer PD printers around that don't use Altos: examples are a 400 spot-per-inch full-color thermal transfer printer and a lower resolution color ink-jet printer. These use a different protocol for receiving files to print; refer to documentation about the PeachPrint command. The exact collection of printers changes over time, so ask around.

Dover printers run Spruce for high volume printing, producing a page per second. one Dover named Clover, is found in room 2106; another, named Menlo, is in room 2305. Samples of the Dover font dictionary may be found next to Clover and Menlo. Instructions for modifying the queue and generally running these Spruce printers are to be found next to their Alto terminals.

In CSL, we have one remaining black and white Press printer, namely Stinger. There is also an experimental Interpress printer, named Quoth. These printers are Ravens (Raven is a Xerox product). The print quality is normally quite good. Instructions for interpreting status displays are posted locally.

Our best quality printer is Platemaker, which is normally operated at 1200 spots per inch, (this can be adjusted, within limits). Platemaker uses a laser to write on photographic paper or film. Color images can be done in individual separations, which are then merged using the Chromalin process. The Platemaker printing process is used for final prints of fine images or for printing masters for publication. The Platemaker needs an expert to run it and to develop the film.

Sending and Receiving Mail

We rely very heavily on an electronic mail system. We use it for mail and also for the type of announcement that might, in other environments, be posted on a physical or electronic bulletin board. In our environment, a physical bulletin board is pretty useless, since people spend too much of their days staring at their terminals and too little wandering the halls. Electronic bulletin boards might work satisfactorily. But a bulletin board, being a shared file to which many people have write access, is a rather tricky thing in a distributed environment. It probably presupposes a distributed transactional file server, for example. Mumble. For whatever reason, the fact remains that we don't have an electronic bulletin board facility at the moment. As a result, announcements of impending meetings, "for sale" notices, and the like are all sent as messages directed at expansive distribution lists. If you don't check your messages once a day or so, you will soon find yourself out of touch (and saddled with a mailbox full of obsolete junk mail). And conversely, if you don't make moves to get on the right distribution lists early, you may miss lots of interesting mail. This business of using the message system for rapid distribution of announcements can get out of hand. One occasionally receives notices of the form: "meeting X will start in 2 minutes—all interested parties please attend".

Grapevine is the PUP-based distributed transport mechanism that delivers the local mail [33]. When talking to Grapevine, individuals are referred to by a two-part name called an "R-name", which consists of a prefix and a registry separated by a dot; for example, "Ramshaw.pa" means Ramshaw of Palo Alto. In addition to delivering the mail, Grapevine also maintains a distributed database of distribution lists. A distribution list is also referred to by an R-name, whose prefix conventionally ends in the character up-arrow, as in "CSL↑.pa". Distribution lists are actually special cases of a construct called a Grapevine "group". Groups can be used for such purposes as controlling access to IFS directories. When you send a message, your mail sending program hands it to any convenient Grapevine server. That server expands any distribution lists. Expanding a distribution list may require contacting another server to access registration data not contained locally. The server then forwards the message to each server that needs a copy. Most users have two servers that can store their mail. If the primary server for a user is down, the message will be sent to that users secondary server. In general, if a server crashes, the mail on it is trapped, but everything else should continue to work reasonably smoothly.

There is a program named Maintain that allows you to query and update the state of the distribution list database. In fact, there are two versions of Maintain: the documented one with the unfortunate teletype-style user interface is used from within Laurel or the Mesa Development Environment <34>: the undocumented one with the futuristic menu interface is used from within Cedar. Some distribution lists are set up so that you may add or remove yourself using Maintain. If you try to add yourself to Foo↑.pa and Maintain won't let you, the proper recourse is to send a message to the distribution list Owners-Foo↑.pa, asking that you please be added to Foo↑.

At the moment, Grapevine pretty much has a monopoly on delivering the mail. But there are several different programs that give users access to Grapevine's facilities from different environments. From an Alto, one uses Laurel [35], which is mentioned elsewhere as a pioneer of modeless editor interfaces. In the Mesa Development Environment, the program Hardy provides services analogous to Laurel's. From within Cedar, most folk use Walnut. Walnut [48] represents a step towards the future in some respects, since Walnut uses Cypress, the Cedar database

management system, to store your mail in a database. Wallaby [49] allows the Walnut user to easily and efficiently pose queries about his mail database, simply by filling in a form. Access to Grapevine from within Cedar can also be had without the database frills through a program called Peanut, which stores your messages in a structured Tioga document instead of in a database. Finally, in case travel should take you away from your multi-function personal workstation, there are servers on the Internet known by the name "Lily" to whom you can connect from any random teletype in order to peruse the mail sitting in your Grapevine mailbox.

Mail systems here understand about Arpanet recipients; mail for these folks is currently sent through a Dorado-based mail gateway to the Arpanet. There is also a mail gateway between Grapevine and the NS world. You can get a message to/from NS land and the Arpanet by going through both gateways. Remote login and file transfer to other Arpanet machines are available through PARCVAX, (or VAXC) a Vax 11/785 running Unix 4.2 BSD.

Packaging Systems and Controlling Versions

History: In the BCPL world, the primary facility for packaging up a group of related files and either handing them out or storing them for later recall was the *dump file*. A dump file, given conventional extension ".dm", is simply the concatenation of the dumpees, together with enough header information to allow the dumpees to be pulled apart again. Dump files have fallen out of favor.

In the Alto/Mesa world, and more strongly, in the Cedar world, a collection of software called "DF files" has grown up that attacks the problem of describing and packaging systems, detailing their interdependencies, and controlling the versions of things. You can find out a lot about DF files by reading Eric Schmidt's dissertation [36]. You can find out how to use DF files by reading the reference manual for DFTool, the Cedar tool that deals with DF files <37>. All that I will try to do here is to give you some idea of what DF files are good for, and how, in a general sense, they are used. One way or another, all Cedar programmers must make their peace with DF files: they perform valuable functions, and they have no current competition.

In the simplest case, a DF file just consists of a list of the names of a related set of files. At this level, a DF file is something like a dump file: given the DF file, you can get at each of the files that it describes. Of course, you want to be sure that you get the right versions of the described files, so just having the DF file list their names isn't quite enough. If there were an Internet-wide notion of version number that made sense, we could get around this problem by specifying the version number along with the file name. But there isn't. The closest thing to a Internet-wide unique identification stamp that we have is the create date of the file. Thus, what a DF file really contains is a list of file names and associated create dates.

The Bringover function of the DFTool will retrieve to your local file system the set of files described by a particular DF file. This would be something of a challenge unless the DF file included some hint to Bringover concerning where in the great, wide Internet the correct versions of these files might be found. So DF files do indeed include such hints: in particular, they include specifications of remote directories on which to look. These directories are just hints, in the sense that Bringover will always verify by checking the create date that it is getting you the correct version of the specified file. If Bringover can't find the correct version on the specified directory, it will issue a sprightly error message. Bringover has lots of bells and whistles. For example, you can point it at either a local or a remote DF file: you can ask it to retrieve just a selected subset of files to your local disk, rather than the entire set described by the DF file; or you can individually consider the files one by one, deciding which you would like to retrieve and which you wouldn't.

Suppose that I am working on a collection of files, such as the sources for this Briefing Blurb. I have made a DF file that describes them, and I can use Bringover to retrieve them from their

remote and permanent home to my local file system, where I can edit them. The next thing that I need is a service that is symmetric to Bringover: after doing my editing, I want to put the new versions back on the remote file server, along with a new DF file that describes the new versions. This function is performed by the SModel button in DFTool. I run SModel, and point it at the old DF file. SModel considers each file in turn, and looks to see if I have edited it: that is, it looks to see if the create date of that file in my local file system is now different than the create date stored in the old DF file. If so, SModel deduces that I have edited the file. It stores the new version that I have made out onto the remote directory. After doing this for each file in turn, SModel writes a new version of the DF file itself, filling in all of the create dates correctly to describe the new version of the entire ensemble. If the DF file describes itself, as most DF files do, SModel is smart enough to make sure that the new version of the DF file is stored out to the remote server as well. SModel also has lots of bells and whistles, but let's not go into them.

If that were the whole story, mere mortals could figure out DF files without straining their brains. But there's more. So far, we have only discussed DF files as descriptions of ensembles of files. In fact, these ensembles are often components of large programs. And this has consequences.

First, there are two distinctly different reasons that you might have for retrieving a program: you might want to change it, or you might just want to run it. In the latter case, you don't need to bring over all of the sources; all that you need is the runnable ".bcd". We could handle this by having two DF files: one for the users and the other for the maintainers. But that would be a disaster: the two DF files would never agree! Instead, each DF file distinguishes between files that it "exports" and the rest. The exported files are the ones that users need, while maintainers are assumed to need the entire ensemble. Bringover can be warned that you are a user rather than a maintainer by specifying Access: public.

Secondly, some programs are going to depend upon other programs: that is, the programs themselves will be "users" ("clients" is a better word here). This suggests that one DF file should be able to contain another DF file. In fact, there should be several different kinds of containment, corresponding to such phrases as:

"They who maintain me also maintain the stuff described by the following DF file."

"They who maintain me are also users (but not maintainers) of the stuff described . . ."

"They who use me are also users of the stuff described . . ."

You get the point? For more details on the ways that these things are done ("includes" and "imports"), check out the reference manual.

In case you still aren't convinced that things are complicated, it is now time to mention the fact that DF files are used for yet another purpose: they describe components of the Cedar release. During the Cedar release process, all of the new versions of Cedar components, which are sitting out on development directories, must be checked for consistency, and then moved en masse to the official release directory. And an entire new set of DF files must be produced, describing the released version of the system (as opposed to the development version). This means, among other things, that some DF files must specify two different remote directories: the development directory and the release directory. In addition, there is a third DFTool function, called Verify, whose purpose is to perform certain consistency and completeness checks on a DF file. By insisting that all component implementors have successfully run Verify on their components, the Cedar Release Master ensures that the release process has at least a fighting chance of succeeding.

DF files grew up over time in response to a mixed bag of needs. As they became more popular, features were added one by one to make them more useful in these varying contexts. The resulting system as a whole is rather hard to grok, but I hope that this introduction has given you a leg up on the problem, at least.

Some Tidbits of Lore

About CSL

CSL has a weekly meeting on Wednesday afternoons called Dealer, starting at 1:15. The name comes from the concept of "dealer's choice"—the dealer sets the ground rules and topic(s) for discussion. When someone says she will "give a Dealer on X", she means that she will discuss X at some future weekly meeting, taking about 15 minutes to do so (plus whatever discussion is generated). Generally, such discussions are informal, and presentations of half-baked ideas are encouraged. The topic under discussion may be long-range, ill-formed, controversial, or all of the above. Comments from the audience are encouraged, indeed, provoked. More formal presentations occur at the Computer Forum on Thursday afternoons: the Forum is not specifically a CSL function, and it is open to all Xerox employees, and sometimes also to outsiders. Dealers are also used for announcements that are not appropriate for distribution by electronic mail. Members of CSL are expected to make a serious effort to attend Dealer.

On occasions of great festivity, Dealer is replaced by a picnic on the hill (that is, Coyote Hill, across Coyote Hill Road), with Mother Xerox picking up the tab.

The CSL Archives (not to be confused with the Cedar Archive System) are a collection of file cabinets and 3-ring binders that provide a continuing record of CSL technical activities. The archives are our primary line of defense in legal matters pertaining to our projects. They also make interesting reading for anyone curious about the history of any particular project.

There is also an institution known as the CSL Notebook, which exists to make all of the potentially interesting documentary output of CSL folk easily accessible to all CSL folk. Trip reports, design documents, immigration manuals (like this one): they should all be submitted to the CSL Notebook <38>. If you thought that it was worth writing down, it is pretty likely that there are other folk in CSL who would consider it worth reading, and submitting it to the CSL Notebook is one easy way to get it read. (I believe that likely looking submissions to the CSL Notebook are considered for entry into the CSL Archives as well.)

Some Code Phrases

You may occasionally hear the following incomprehensible phrases used in discussions, sometimes accompanied by laughter. To keep you from feeling left out, we offer the following translations:

"Committing error 33"

(1) Predicating one research effort upon the success of another. (2) Allowing your own research effort to be placed on the critical path of some other project (be it a research effort or not). Known elsewhere as Forgie's principle.

"You can tell the pioneers by the arrows in their backs."

Essentially self-explanatory. Usually applied to the bold souls who attempt to use brand-new software systems, or to use older software systems in clever, novel, and therefore unanticipated ways ... with predictable consequences. Also heard with "asses" replacing "backs".

"We're having a printing discussion."

Refers to a protracted, low-level, time-consuming, generally pointless discussion of something peripherally interesting to all. Historically, printing discussions were of far greater importance than they are now. You can see why when you consider that printing was once done by carrying magnetic tapes from Maxc to a Nova that ran an XGP.

Fontology

The body of knowledge dealing with the construction and use of new fonts. It has been said that fontology recapitulates file-ogeny.

"What you see is what you get."

Used specifically in reference to the treatment of visual images by various systems, e.g., a Bravo screen display should be as close as possible to the hardcopy version of the same text. Also known is some circles by the acronym "WYSIWYG", pronounced "whiz-ee-wig".

"Moving right along", *"Pop!"*, or *"Hey guys, up-level!"*

Each of these phrases means that the conversation has degenerated in some respect, often by becoming enmeshed in nitty-gritty details. Feel free to shout out one or more of these phrases if you feel that a printing discussion has been going on long enough. If two participants in a large meeting begin discussing details that are of interest to them but not of interest to the group as a whole, shout *"Off-line!"* instead.

"Life is hard"

Two possible interpretations: (1) "While your suggestion may have some merit, I will behave as though I hadn't heard it." (2) "While your suggestion has obvious merit, equally obvious circumstances prevent it from being seriously considered." The charm of this phrase lies precisely in this subtle but important ambiguity.

"What's a spline?"

"You have just used a term that I've heard for a year and a half, and I feel I should know, but don't. My curiosity has finally overcome my guilt." Moral: don't hesitate to ask questions, even if they seem obvious.

Hints for Gracious Living

There are a couple of areas where life at PARC can be made more pleasant if everyone is polite and thoughtful enough to go to some effort to help out. Here are a few words to the wise:

Coffee

CSL has 2 coffee alcoves where tea, cocoa, and several kinds of coffee are available. All coffee drinkers (not just the secretaries or some other such barbarism) help out by making coffee. If you are about to consume enough coffee that you would leave less than a full cup in the pot, it is your

responsibility to make a fresh pot, following the posted instructions. There are lots of coffee fanatics around, and they get irritated beyond all reason if the coffee situation isn't working out smoothly. For those coffees for which beans are freshly ground, the local custom is to pipeline grinding and brewing. That is, you are expected to grind a cup of beans while brewing a pot of coffee from the previous load of ground beans. This speeds up the brewing process for everyone, since a load of ground beans is—at least, had better be—always ready when the coffee pot runs out.

Sharing Office Space

Be warned as well that some lab members are unbelievably picky about the state of their offices. The convention is that any Alto in an empty office is fair game to be borrowed, but most of these have long since benn borrowed and never returned. Private Dandelions and Dorados may be borrowed only by prior arrangement with their owners, because of the potential problems of sharing disk space. Many Dorado owners live cleanly enough that they will be more than willing to let you use their machine when they are not there, but ask first, just to be safe. When people are on vacation, they are supposed to leave notice on one of the designated boards around the lab to indicate whether or not their machine is up for grabs. If you use someone's office for any reason, take care to put everything back *exactly* the way it was. Don't spill crumbs around, or leave your half-empty cocoa cup on the desk, or forget to put the machine back in the state that you found it, or whatever. Of course, lots of people wouldn't mind even if you were less than fanatically careful. But some people do mind, and there is no point in irritating people unnecessarily.

Sharing printers

When you pick up your output from a printer, it is considered antisocial merely to lift your pages off the top of the output hopper, and leave the rest there. Take a moment to sort the output into the labelled bins. Sorting output is the responsibility of everyone who prints, just as making coffee is the responsibility of everyone who drinks (coffee). Check carefully to make sure that you catch every break page: short outputs have a way of going unnoticed, and hence being missorted, especially when they come out right next to a long output in the stack. The rule for determining which bin is to use the first letter that appears in the name on the break page. Thus, "Ramshaw, Lyle" should be sorted under "R", while "Lyle Ramshaw" should be sorted under "L". A trickier question is what to do with output for "Noname", or the like. Following the rule would suggest filing such output under "N", but that doesn't seem very helpful, since the originator probably won't find it. Check the contents and file it in the right box if you happen to recognize whose output it is; otherwise, either leave it on top of the printer or stick it back in the output hopper.

The phone system

When the Voice Project has had its way, our phone system will be a marvelous assemblage of computers talking to computers. Presently, many folks in CSL have etherphones, funny little boxes that play tunes rather than ringing like normal phones. All etherphone-to-etherphone conversations are carried as packetized voice on an ethernet. Other phone calls simply make use of Pacific Bell's services, so there isn't too much to say.

First, a little preaching. If you make a significant number of personal long-distance phone calls from Xerox phones, it is your responsibility to arrange to reimburse Xerox for them. This may not be that easy, either, since phone bills take quite a while (six weeks or so) to percolate through

the bureaucracy upstairs, and the said bureaucracy also has a lot of trouble figuring out where to send the phone bills of new people, and people who move around a lot. Just because it is easy to steal phone service from Xerox doesn't make it morally right; if you think you aren't being paid enough, you should start agitating for a raise. If enough suspicious calls are made without restitution, PARC (being a bureaucracy) will impose some bureaucratic "solution" on all of us.

So as not to end on a sour note, let's discuss how the phone system works, anyway. The offices within PARC have four-digit extensions within the 494 exchange, a system known as Centrex; to dial another office, those four digits suffice. Dialing a single 9 as the first digit gives you an outside line, and you are now a normal customer of Ma Bell: see a phone book for more details (Oh, come now, surely you know about phone books!). Dialing a single 8 gives you different sounding dial tone, and puts you onto the IntelNet (not to be confused with the InterNet). The IntelNet is a Xerox-wide company phone system, complete with its own phone book, and its own phone numbers. If you are calling someone in some remote part of Xerox, you can save Mother Xerox some bread by using the IntelNet instead of going straight out over Ma Bell's lines. On the other hand, you may not get as good a circuit to talk over—although this situation is frequently said to be improving. Furthermore, through the wonders of modern electronics, you can dial any long-distance number over the IntelNet. Just use the normal area code and Ma Bell number: the circuitry is smart enough to take you as far as possible towards your destination along IntelNet wires, and then switch you over to Ma Bell lines for the rest of the trip. Using the IntelNet doesn't start to save money until the call is going a fair distance: therefore, the IntelNet doesn't let you call outside numbers in area codes 408, 415, and 916—better to just dial 9.

One more thing: after you have dialed a number on the IntelNet, you will hear a funny little beeping. At that point, you are being asked to key in a four-digit number to which the call should be billed. You should use the four-digit extension number for your normal office phone under most circumstances. Calls made by dialing 9 instead of 8 are always charged to the phone from which they are placed.

The first three rings (roughly speaking) of an incoming call occur only in your office. The next roughly three rings happen both at your office phone and at a receptionist's phone, centrally located in the laboratory. During normal business hours, the receptionist's phones are staffed: thus, someone will at least take a message for you, and leave it on a little slip of paper in your physical message box. If the second three rings go by without either of those two phones answering, the call is then forwarded to the guards desk downstairs (I believe).

If you are expecting a call but won't be near your normal phone, a call forwarding facility exists: dial 106 and then the number to which you want your calls to be forwarded. Later on (*try* not to forget), you dial 107 on your normal phone to cancel the forwarding. When I forward my phone, I turn the phone around physically, so that the touch-pad faces the wall. This helps me to remember to cancel the forwarding again later, at which point I turn the phone back the normal way. There is also a way to transfer incoming calls to a different Xerox number: Depress the switch hook once, and dial the destination number; when the destination answers, you will be talking to the destination but the original caller won't be able to hear your conversation; depressing the switch hook again puts all three of you on the line; then you can hang up when you please. If the destination doesn't answer, depressing the switch hook once again will flush the annoying ringing or busy signal.

References

Reference numbers in [square brackets] are for conventional hardcopy documents. Many of them are Xerox reports published in blue and white covers; the CSL blue-and-whites are available

on bookshelves in the CSL Alcove. Reference numbers in <angle brackets> are for on-line documents. The path name for such files is given herein in the form

[FileServer]<Directory>SubDirectory>FileName.Extension

for backward compatibility with earlier systems. Recently, the simpler alternative form

/FileServer/Directory/SubDirectory/FileName.Extension

has begun to come into local currency, but some systems still demand brackets rather than slashes. Much of the Cedar documentation is stored on-line in the directory

[Indigo]<Cedar®>Documentation>

where ® represents the number of the current release.

<*n*>: The generic form for a reference to an on-line document.

[*n*]: The generic form for a reference to a hardcopy document.

[1]: **Sunset New Western Garden Book.** Lane Publishing Company, Menlo Park, CA, 1979. The definitive document on Western gardening for non-botanists; 1200 plant identification drawings; comprehensive Western plant encyclopedia; zoned for all Western climates; plant selection guide in color.

[2]: John E. Warnock. **The Display of Characters Using Gray Level Sample Arrays.** blue-and-white report CSL-80-6.

[3]: Richard F. Lyon. **The Optical Mouse, and an Architectural Methodology for Smart Digital Sensors.** blue-and-white report VLSI-81-1.

[4]: **The Ethernet Local Network: Three Reports.** blue-and-white report CSL-80-2.

[5]: John F. Shoch, Yogen K. Dalal, Ronald C. Crane, and David D. Redell. **Evolution of the Ethernet Local Computer Network.** blue-and-white report OPD-T8102.

<6>: *no reference.*

[7]: David R. Boggs, John F. Shoch, Edward A. Taft, and Robert M. Metcalfe. **Pup: An Internetwork Architecture.** blue-and-white report CSL-79-10.

[8]: **Internet Transport Protocols.** Xerox System Integration Standard report XSIS 028112, December 1981.

[9]: **Courier: The Remote Procedure Call Protocol.** Xerox System Integration Standard report XSIS 038112, December 1981.

[10]: C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs. **Alto: A personal computer.** blue-and-white report CSL-79-11.

<11>: [Indigo]<AltoDocs>AltoHardware.press. Everything that you need to know to write your own Alto microcode.

[12]: **The Dorado: A High-Performance Personal Computer; Three Papers.** blue-and-white report CSL-81-1.

<13>: [Indigo]<DoradoDocs>DoradoBooting.press. Describes how to boot a Dorado, and how to configure it to boot in various ways.

[14]: Myer, T. H. and Barnaby, J. R. **TENEX Executive Language Manual for Users.** Available from Arpa Network Information Center as NIC 16874, but in the relatively unlikely event that you need one, borrow one from a Tenex wizard.

<15>: [Indigo]<AltoDocs>BCPL.press The reference manual for the BCPL programming language.

<16>: [Indigo]<AltoDocs>OS.press. The programmer's reference manual for the Alto

Operating System, including detailed information on the services provided and the interface requirements.

<17>: [Indigo]<AltoDocs>Packages.press. A catalogue giving documentation for the various BCPL packages that other hacker's have made available.

[18]: **Mesa Language Manual, Version 11.0**, March 1984. Published by the Systems Development Department of **OSD**.

[19]: James G. Mitchell, William Maybury, and Richard Sweet. **Mesa Language Manual, Version 5.0.** blue-and-white report CSL-79-3. A cross between a tutorial and a reference manual, though much closer to the latter than the former.

[20]: Morris, J. H. **The Elements of Mesa Style.** Xerox PARC Internal Report, June 1976. Somewhat out of date (since Mesa has changed under it), but a readable introduction to some useful program structuring techniques in Mesa.

[21]: Adele Goldberg and David Robson. **Smalltalk-80: The Language and Its Implementation**. book published by Addison-Wesley, 1983.

[22]: **Interlisp Reference Manual**. Published in a blue and white cover, although not officially a blue-and-white. October, 1983.

[23]: The Interlisp-D Group. **Papers on Interlisp-D**. blue-and-white report CIS-5 (also given the number SSL-80-4), Revised version, July 1981.

[24]: L. Peter Deutsch and Edward A. Taft, editors. **Requirements for an Experimental Programming Environment**. blue-and-white report CSL-80-10.

<25>: [Cyan]<CedarChest®>Top>Documentation.df. Hardcopies are entitled **The Cedar Manual**.

[26]: **Alto User's Handbook**. Internal report, published in a black cover. The version of September, 1979 is identical to the version of November, 1978 except for the date on the cover and title page. Includes sections on Bravo, Laurel, FTP, Draw, Markup, and Neptune

<27>: [Indigo]<AltoDocs>SubSystems.press. Documentation on individual Alto subsystems, collected in a single file. Individual systems are documented on [Indigo]<AltoDocs>systemname.TTY, and these files are sometimes more recent than SubSystems.press.

[28]: Jerome, Suzan. **Bravo Course Outline.** Internal report, published in a red cover. Oriented to non-programmers.

<29>: [Cyan]<Cedar®>Documentation>TiogaDoc.tioga. How to use the Tioga editor.

<30>: [Indigo]<PrintingDocs>PressFormat.press. Describes the Press print file format.

<31>: [Indigo]<PrintingDocs>PressOps.press. Describes the Press printing program.

<32>: [Indigo]<PrintingDocs>PDPrintOps.press. Describes the PDPrint printing program.

[33]: Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. **Grapevine: Two Papers and a Report.** blue-and-white report CSL-83-12.

<34>: [Ivy]<Laurel>Maintain.press. Documentation for the teletype version of Maintain, the version that is used from within Laurel or Tajo.

[35]: Douglas K. Brotz. **The Laurel Manual.** blue-and-white report CSL-81-6.

[36]: Eric Emerson Schmidt. **Controlling Large Software Development in a Distributed Environment.** blue-and-white report CSL-82-7.

<37>: [Cyan]<Cedar®>Documentation>DFToolDoc.tioga. The reference manual for the use of DF files.

<38>: [Indigo]<CSL-Notebook>Docs>HowToUseCSLNotebook.press.

<39>: [Cyan]<CedarChest®>Documentation>HowToUseAPublicCedarMachine.tioga.

[40]: Warren Teitelman. **The Cedar Programming Environment: A Midterm Report and Examination.** blue-and-white report CSL-83-11.

[41]: Mark R. Brown, Karen Kolling, and Edward A. Taft. **The Alpine File System.** blue-and-white report CSL-84-4.

[42]: Dan Swinehart, Polle Zellweger, Rick Beach, Bob Hagmann. **A Structural View of the Cedar Programming Environment.** blue-and-white report CSL-86-1.

[43]: Bob Hagmann. **A Labeless File System For Cedar.** [Indigo]<CSL-Notebook>Docs>86CSLN-0047.press

[44]: Dennis Arnon and Carl Waldspurger. **CaminoReal.** [Cyan]<CedarChest®>Documentation>CaminoRealDoc.tioga.

[45]: Eric Bier and Ken Pier. **The Gargoyle Reference Manual.** [Cyan]<CedarChest®>Documentation>GargoyleDoc.tioga.

[46]: Eric Bier and Ken Pier. **Gargoyle Tutorial.** [Cyan]<CedarChest®>Documentation>GargoyleTutorialDoc.tioga.

[47]: Stephen Ades and Daniel Swinehart. **Voice Annotation and Editing in a Workstation Environment.** blue-and-white report CSL-86-3.

[48]: James Donahue and Willie-Sue Orr. **Walnut: Storing Electronic Mail in a Database.** blue-and-white report CSL-85-9.

[49]: Jack Kent and Doug Terry. **Wallaby.** [Cyan]<CedarChest®>Documentation>WallabyDoc.tioga.

[50]: Jean-Marc Frailong. **XTSetter.** [Cyan]<CedarChest®>Documentation>XTSetterDoc.tioga.

[51]: Abbay Bhushan and Michael Plass. **The Interpress Page and Document Description Language.** IEEE Computer, Vol 19, Number 6, June 1986, pp. 72-77.