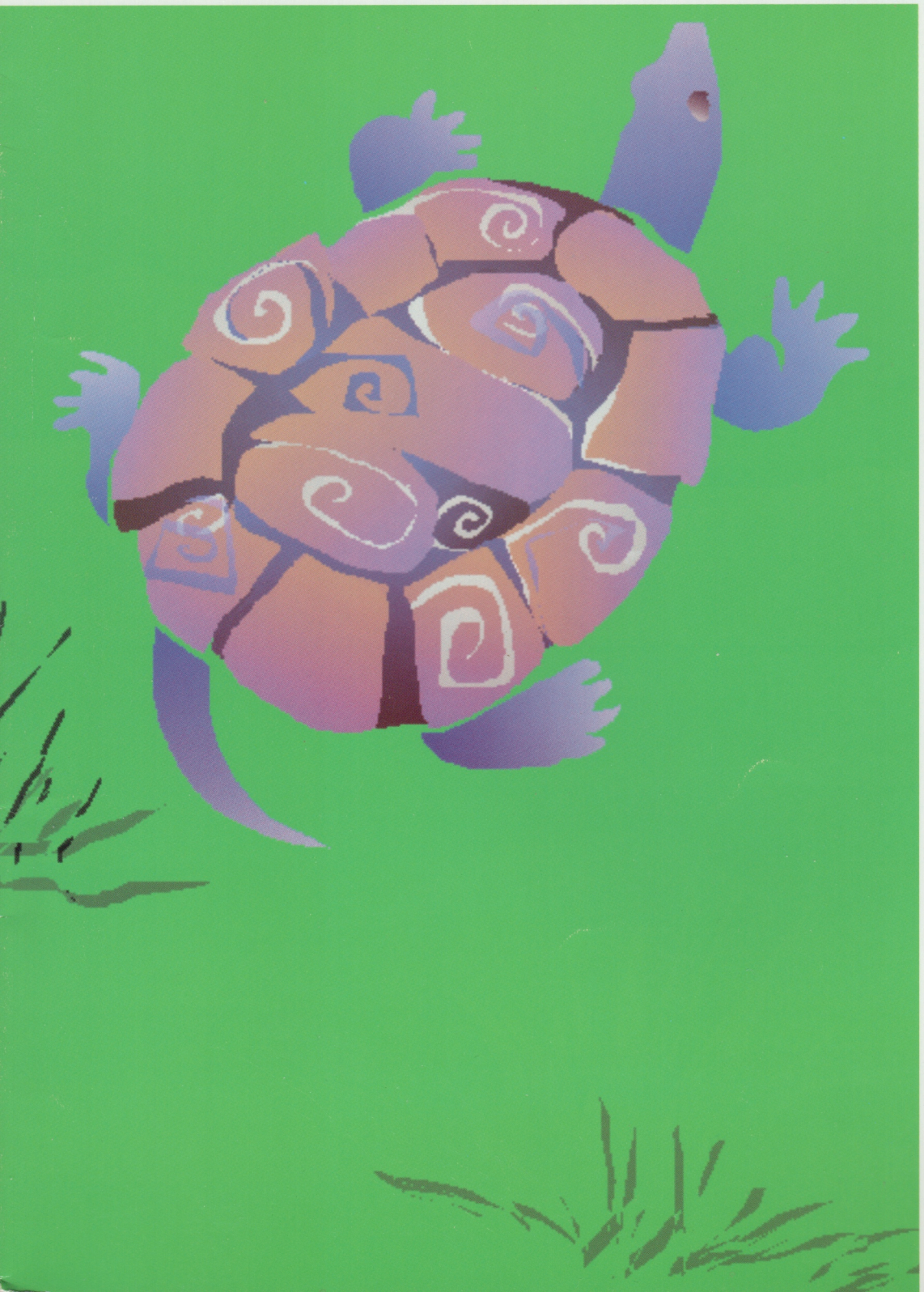
develop

The Apple Technical Journal



Issue 3 July 1990

EDITORIAL

Editor in Chief's Clothing *Louella Pizzuti*
Chief Engineer *Dave Johnson*
Managing Editor *Carol Westberg*
Developmental Editors *Lorraine Anderson, Ann Cullen*
Copyeditors *Toni Haskell, Caroline Sloan*
Editorial Assistant *Lenore Zelony*
Indexer *Ira Kleinberg*
Manager, Developer Press *David Krathwohl*

ART & PRODUCTION

Design/Art Direction *Joss Parsey*
Technical Illustration *J. Goldstein*
Production *Bruce Potterton*
Printing *Craftsman Press*
Film Preparation *FilmCraft*
Photographer *Ralph Portillo*
Animation *Hal Rucker*
Circulation Management *Dee Kiamy*
Online Production *Cassi Carpenter*

REVIEW BOARD

Pete "Luke" Alexander
Tim "Sweetpea" Enwall
Larry "Cat Couch" Rosenstein
Andy "The Shebanator" Shebanow

SPECIAL THANKS

Mark Baumwell
Jim Luther

Cleo Huggins created the cover illustration in PixelPaint. Special thanks to Charlie's turtle Rodney for modeling.

d e v e l o p, *The Apple Technical Journal,* is a quarterly publication of Developer Press.

CONTENTS

LOUELLA PIZZUTI

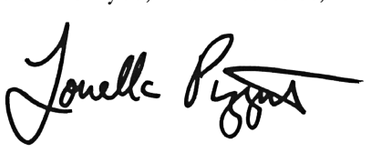
Dear Readers,

By now, I'm tiptoeing through the bulb fields of Holland and painting like a fiend (note the backpack and the tulips in the photo). But because of Carol, Dave, the review board, the technical reviewers, this issue's authors, and Lenore directing a host of others, you are holding the third issue of develop.

Carol "Editor in Louella's Clothing" Westberg is the managing editor of develop while I'm out on my much-needed sabbatical; she's the one who's making sure that the articles say what we want to say (and hopefully what you want to hear) in a way that we'd like to say it. Dave "if you can hold it, I can juggle it" Johnson joined us in our develop adventures just two short weeks before I took off. As our staff engineer, the technical buck stops with him. He makes sure that all of the code is fully tested and that technical disagreements end in decisions, not fist fights. The review board is a group of talented, thorough, dedicated, and vocal Apple engineers who ride herd on the rest of us. Together, Andy Shebanow, Pete Alexander, Tim Enwall, and Larry Rosenstein steer develop (and the articles within each issue) in the right direction.

The review board guides us on the content, but when it comes to covers, Cleo (see the pages describing the *Developer Essentials* disc for her bio) and I go wild. So, why a tortoise? Well, this issue's theme is CD-ROM and when confronted with CD-ROMs, many people first think "slow!" It's true, that compared with a hard disk, a CD-ROM disc can seem to move like molasses on a winter morning. Or like a tortoise in a race with a hare. Hmmm, how did that fable go? Seems to me that everyone expected the hare to win hands (or feet) down. But initial impressions notwithstanding, the tortoise took off and came out the clear winner. The same applies to CD-ROM; slower than a hard disk, sure, but faster by far than a human. In the end, the information and the space are well worth the wait.

Take a look at the CD-ROM articles and figure out how you, like the tortoise, can end up a technological winner.



Louella Pizzuti
Editor

LOUELLA PIZZUTI is a career woman in shorts who freely admits that her favorite joke isn't funny. Her favorite part of working at Apple is the texture, and her favorite place to be is outside.

Inside or outside, her favorite color's yellow (the color of madness) and her favorite painter's Vince.

She likes to play: in the mud (usually planting flowers); on the soccer field; in the weeds (it's easier than pulling them); but never, ever with words.

She's got her lemon tree (finally) and the convertible is next. We're all wondering what will happen to the flowers in the bed of her truck. •

LETTERS

I just wanted to let you know that I have found both issues of develop to be extremely useful. You do an excellent job in picking up where *Inside Macintosh* leaves off. I hope Apple continues to publish develop for a long time.

— Paul Higinbotham

Of all the technical journals I receive about the Macintosh, Digital VAX, and computing in general, all pale in comparison to your develop. Please send me a copy.

—Steve Salika

In your last issue, Curt Bianchi tells me to beware of how the Memory Manager grabs my Pascal object's handles. Then, in another article, Richard Clark tells me that the Memory Manager has a secret life and that I should be careful when I pass it pointers. My question is, should I ask the Memory Manager to take a blood test before we get serious?

—Concerned in Palo Alto

Although you are right to be cautious about getting serious with the Memory Manager, there is really no need for a blood test. The very articles that you mention present clear guidelines for having safe yet fulfilling, uh, "interactions" with the Memory Manager. These guidelines, judiciously and conscientiously (and enthusiastically!) followed, provide all the protection you will need.

—Dave Johnson

After using both on-line documentation and hard copy for a while, I prefer hard copy. It is easier to read, and more immediate. I like the concept of having complex and interrelated documentation on-line with cross referencing a click away. When you obtain instant response time, on-line documentation will be invaluable.

—Thomas Bernard

STEPS FORWARD

Thanks for all the fantastic work on develop —I love it, and being an American abandoned in England, it is one of the most informative journals I get over here. I do have a couple of gripes (read, "winges" in England) about the software used to display the articles.

The control panel windoid is a complete pain—it doesn't fit on a 13-inch monitor without obscuring the text window unless you move it halfway off the screen, so you have to be continually moving the panel around the screen. I then decided to print out some of the articles, but that isn't much easier, because you either have to know the page where each article begins, or navigate there one page at a time.

It would be nice to see a control panel which is vertically oriented so that it can fit along with the text window on a 13-inch screen, and to also include a button which jumps to the next article—moving to the first and last pages is not really that useful. I think it would also be much more useful if the

COMMENTS

We welcome timely letters to the editor, especially from readers wishing to react to articles that we publish in develop. Letters should be addressed to Louella Pizzuti, 20525 Mariani Ave., M/S 75-3B, Cupertino, CA 95014 (AppleLink Pizzuti1). All letters should include name and company name as well as address and phone number. Letters may be excerpted or edited for clarity and space.

SUBSCRIPTION INFORMATION

Use the order form on the last page of the journal. Please address all subscription (and subscription-related) inquiries to

develop
Apple Computer, Inc.
P.O. Box 531
Mt. Morris, IL 61054 U.S.A.
AppleLink Address: DEV.SUBS

text for each page fit in a window rather than having to scroll each window as well as forward each page.

.

—David L'Heureux

As you know, we're entering an era in the Macintosh world that revolves around electronic publishing. Many of the standards and interfaces that are so well defined in the Macintosh desktop metaphor don't exist when you're creating interactive electronic magazines (Hyperzines). Developer Essentials *is a living document. It will continually grow and change as we begin to determine what works (and what doesn't)—things like how you use sound, how and when you should animate an icon, what's the best use of color, where is the best place within the virtual magazine metaphor to put a control panel windoid and have it not be a complete pain. Ya know, things like that.*

The reason we've put the automatic feedback capabilities into develop *is to get your ideas, criticisms, and thoughts. We don't have all the answers and will be experimenting with new ways of representing data. We hope to have more defined and stable human interface standards regarding the use of electronic media like CD-ROM in the near future. In the meantime, look for experimentation, a few mistakes, and some very open minds looking to you for feedback and suggestions here at Apple.*

—Scott Converse
Electronic Media Group Manager

Issue 2 of develop is great. The articles are fun, informative, and well written. I think you are off to a great start.

But I must strenuously object to your rampant waste of paper for the sake of "design." Almost every page has at least 25 percent white space; many have more. I don't know if you have an aversion to trees or just a lack of concern for our children, but your choice in this matter does not reflect Apple's generally ecological view. Apple is a leader in reducing waste in manufacturing, but you insist on creating waste in your magazine.

Please, please take a look at redesigning your pages for future issues. Apple's publications are often very well designed, but yours is the only one that screams "paper waste" on every page. It is an easy step that will help the whole world.

—Paul Hoffman

I'm looking into using recycled paper for develop *(in fact, it started out as a requirement for the first issue), but I've run into conflicting information. I recycle my paper (both here at work and at home) and am actively looking for ways to help* develop *fit into the ecological scheme of things. Our printer recycles all of the waste (generated from printer make-ready, and overages), and the paper we print on, like most paper these days, has a recycled component. Some people I've spoken to advocate recycled paper as the answer to all of our problems; others contend that the chemicals used to de-ink the paper damage the environment more*

than they help to save it. If you've got ideas about whom I could talk to to hear the real scoop on recycled paper (from environmental impact to lasting qualities), I'd love to hear them.

Meanwhile the page design is intended to leave room for notes (which many developers have told me they make), and for readability. The column widths must allow for full-page width code listings but must also work with readable line lengths. I'm sorry that it screams paper waste to you, and I will talk to our designer about ways in which we might adapt the design.

—Louella Pizzuti

STEPS BACK

When we find technical errors in previous issues of develop (or when you point them out to us), we make corrections in the text and code for the current *Developer Essentials* disc. You can also find corrections in this section of the journal.

So far, we want to let you know about these changes:

On page 75, the abstract should read "Through the Slot Manager system software, the Macintosh can read the declaration ROMs in NuBus slots and processor slots, like those in the Macintosh SE/30. This article tells you what you must know about NuBus addressing and the structure of correct declaration ROMs to successfully debug the ROM. It walks you through the structure of an example declaration ROM and gives common errors and strategies for debugging declaration ROMs."

On page 91, "Assuming the board is in slot $B, the above format block (residing on byte lane 3)" should be byte lane 0.

On page 149, the procedure **`MyVScrollCallback`** appears twice. The second one should have been **`MyHScrollCallback`**, as indicated in the comments. Thanks to Sam Roberts for pointing this one out. For those of you who didn't even notice, shame on you.

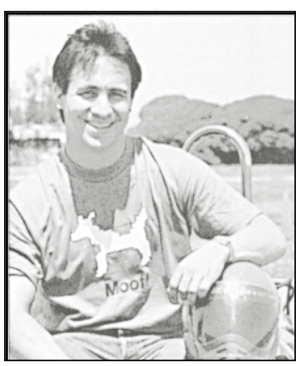
There was a pair of bugs in the "Heap Demo" source code distributed with Issue 2 of develop, one of which prevented the source code from compiling. The Developer Essentials *disc contains a corrected version (HeapDemo 1.3.4) In order to compile the old code, you should remove the reference to "UMonitor" at the start of HeapDemo.p. The UMonitor unit is a debugging tool that was used during the final checkout of the Heap Demo, and though I removed the calls to its code, I forgot to remove the USES reference.*

The other bug was in the menu enabling logic. With the bug, you could crash the program by closing the memory window, and deleting all blocks. The menu enabling logic has been changed to fix this problem. Thanks to the diligent readers who pointed out these problems. I'll make sure you have more articles to nit-pick in the future!

—Richard Clark

CD-ROM: THE CUTTING EDGE

Just as applications changed when hard disks became widely available, they are beginning to change again to take advantage of the newest storage technology to go mainstream: CD-ROM. The acronym, which stands for Compact Disc—Read Only Memory, doesn't begin to tell how the technology can liberate your applications from the currently established limitations of magnetic media. This article discusses the ups and downs of CD-ROM and looks at simple and creative ways developers can take advantage of this new medium.



MARK B. JOHNSON

With its advantages over traditional magnetic media, CD-ROM can help you establish a real difference between your products and those of your competitors. It can enable you to produce ground-breaking applications at an affordable cost, unconstrained by disk space limitations. Although it does have its own limitations, its advantages and the possibilities they offer should be enough to convince you to make the investment in this competitive weapon.

THE CD-ROM EDGE

CD-ROM puts traditional magnetic media to shame in more ways than one. Beyond the obvious advantages of high capacity and low cost, no magnetic media can match CD-ROM in the areas of versatility, durability, portability, and interchangeability.

Capacity. Optical encoding of digital data enables storage of up to 660 MB of information on a single 120-mm CD-ROM disc. It would take more than 800 floppy disks to hold the same amount of information. Although some recording techniques limit CD-ROM discs to 550 MB, even a disc with this lower limit holds more information than hundreds of floppy disks.

Economy. You would think that with its capacity and durability, CD-ROM would be quite expensive. On the contrary, when you need to distribute at least 10 MB of data

MARK B. JOHNSON (emphasis on the "B") has been driving people crazy at Apple and DTS for two years. This Hoosier native came here from Notre Dame, where he spent three and a half years in the snow, getting for his trouble only a 1986 B.A. in French and computer applications (a CS/MIS sort of thing) with a concentration in philosophy. However, he liked the seasons there (and the White Castle hamburgers) so much that he convinced them to let him stay and slave for another three years doing Apple support and development in user services. Mark does a lot of the "unofficial" projects in DTS. He has had a hand in *Phil & Dave's Excellent CD™*, *SpInside*

to 100 or more people, CD-ROM is a very cost-effective medium. Large numbers of discs (over 1,000 units) can be produced for near two dollars per disc, almost as low a cost as for producing a standard 800K floppy disk.

Versatility. A CD-ROM disc can contain both digital data and high-quality digital audio tracks (usually copied directly from a digital audio tape). Thus, this technology opens the door to new combinations of data and sound on a single medium. See Eric Mueller's article "Accessing CD-ROM Audio Tracks From Your Application" in this issue for insight into how such things are done.

Durability. Because CD-ROM is a read-only medium, a disc has a high degree of data integrity once it is pressed. Users cannot erase or overwrite the data, nor can they accidentally infect the disc with the latest virus. CD-ROM discs are mostly comprised of, and completely coated by, durable plastic. This construction, combined with the fact that the data encoding method used today provides its own error correction, makes CD-ROM a highly stable and durable medium, resistant to the handling and magnetic damage that commonly affect other types of media. (To prove this point a few years ago, an Apple engineer buried some CD-ROM discs in his cat's litter box for a week. He then wiped the discs clean and successfully read all the data from them.)

Portability. Another advantage of optical technology is its portability. Unlike most sealed hard disk platters, CD-ROM discs can be removed from a CD-ROM drive. This feature gives the user a virtually unlimited amount of storage space without locking him or her into a single type of information from minute to minute.

Interchangeability. Any CD-ROM player can read any CD-ROM disc at the bit level, since all CD-ROM discs, whether they contain data, audio, or both, share a common physical format. To go one step farther, developers can use a standard file system format to enable users to access a disc on a variety of hardware configurations under several different operating systems. Brian Bechtel's article "The Ins and Outs of ISO 9660 and High Sierra" in this issue explores the two standard file system formats. Developers can also mix partitions on a disc so that it can be read by both the Macintosh® operating system and ProDOS®, as explained in Llew Roberts's article "How to Create a Mixed-Partition CD-ROM" in this issue.

SO WHAT'S THE CATCH?

All this for one low price? There must be a catch. Not exactly. Some minor limitations, yes—in the areas of speed, inability to write on the medium, and cost of a CD-ROM drive—but no show-stoppers.

Speed. Due to the mass of the optical read head and the data encoding methods used, reading from a CD-ROM disc is slower than reading from a good hard disk, but still faster than reading from a floppy. The speed of reading from disc, which is

Macintosh, and the Apple FTP Internet site. He also stays up nights figuring out how to best hide Technical Note #31.

In his rare moments away from the office, he loves watching Notre Dame beat USC in football, participating in a variety of sports, collecting French wine, playing in his recently rediscovered kitchen, and working on and driving fast cars (very useful, since he is rumored to own a 1975 Bricklin SV-1). Mark claims to be addicted to adrenaline and electronic mail, but from the 550 (and counting) Coke cans stacked in his office window, we know what his *real* addictions are.

150K per second, is sufficient for just about anything but uncompressed, full-motion color video, but any application that requires a lot of disc access (for example, HyperCard) suffers some performance degradation if run directly from CD-ROM instead of from a hard disk. Of course, if you are just using CD-ROM as a means of distributing your software, then speed really is not a problem since users are most likely going to copy your software to their hard disks or file servers to use. But even in cases where speed does matter, there are things you can do to address this concern; see the sidebar "Maximizing CD-ROM Speed."

MAXIMIZING CD-ROM SPEED

If you are writing CD-ROM-based software, there are many steps you can take to ensure the maximum possible speed from your product.

First, organize your files in a way that makes it easy for both the user and the Finder to work. Avoid burying important files six or seven folders from the root level of the file system, and conversely, avoid putting 1,000 files into a single folder or at the root level. Once you have the master hard disk organized, look for opportunities to precompute and store information to speed up your application. Take advantage of the 660 MB of space at your disposal—it is less expensive than CPU cycles for your users.

Take full advantage of the static nature of CD-ROM; if it helps, use absolute filenames in your applications or HyperCard stacks if they are to be run only on the disc. However, note that if you decide to use absolute filenames, you should provide a mechanism such as a preferences file in the user's Preferences folder to enable users to change the location of your applications and data. One method for doing this is hard-coding only default pathnames and prompting the user (saving the resulting choices to a preferences file) if the files the user needs are not found in the default locations.

The best method for gaining speed on the CD-ROM is to thoroughly test the entire contents on different machine configurations until you find the optimal configuration. Then go back and test it again to make sure you have not missed anything. Different layouts, default memory configurations, and pathnames can produce different results, especially for HyperCard® stacks. If possible, test the hard disk as a locked volume on a file server or on a CD-ROM simulator (usually available at your local disc mastering company); these tests give a more accurate reflection of how the final CD-ROM should perform than testing on a locally connected hard disk.

Finally, before you master the CD-ROM, optimize the hard disk that you will use for the mastering process. You can either use a third-party optimization program (be sure to back up your disk first) or even better, copy the entire contents to a freshly formatted hard disk. This process guarantees that your files are written (and will be read) contiguously on the disk and also rebuilds the desktop resource file or desktop database files that the Finder needs. When you finish this process, unmount the hard disk and use a disk editor to zero the boot blocks on the hard disk. This final process speeds the time it takes the CD-ROM to mount and also ensures that it is not mistaken for a boot volume.

If you do these things, you will have done everything you can to make your CD-ROM product as fast as possible.

Inability to write. The read-only feature that makes CD-ROM so durable is considered by some to be a disadvantage to this technology. Since you cannot erase or overwrite CD-ROM, it is not a replacement for magnetic media. Optical technologies that allow writing as well as reading are coming to the market—including read-only and erasable-optical discs, laser videodiscs, and WORM (write once, read many) discs—but none of these can currently match the cost-effectiveness of CD-ROM for electronic publishing and distribution.

Cost of a drive. Although Apple recently cut the price of the AppleCD SC to be comparable to that of an average-sized hard disk, many developers still consider the cost prohibitive for the mainstream user, and therefore do not consider a CD-ROM drive a mainstream peripheral. It is true that today's installed base of CD-ROM drives is small, but it is beginning to grow at a faster rate as more and more new titles appear on the market. Here we have the familiar chicken-or-egg problem: developers want to wait for the installed base to reach a certain point before they develop for CD-ROM, and users want to wait for the number of useful CD-ROM products to reach a certain point before they buy a CD-ROM drive.

Still, the fact is that it may only take a single blockbuster CD-ROM to motivate many people to buy. For example, when Apple's Developer Technical Support group produced the original *Phil & Dave's Excellent CD* (now a collector's item), many developers found it reason enough to buy a CD-ROM drive. If your product is good enough, people will purchase the necessary hardware to use it. And remember that competition is forcing the cost of CD-ROM drives and discs ever downward, so you can bet that by the time you get your product to market, price will be less of a limiting factor than it is now.

A WORLD OF POSSIBILITIES

Thanks to its advantages over other technologies and despite its minor limitations, CD-ROM opens up a world of new possibilities to the developer. CD-ROM is especially suited to

- distributing large products
- creating new or enhanced versions of products
- distributing application environments
- enabling collaborative products
- distributing information products
- enabling interactive media products
- enabling user interaction in a learning environment

We'll look at some examples of each of these kinds of uses.

As we look at these examples, keep in mind that although building a CD-ROM is essentially as easy as cloning a hard disk, building in a look and feel that takes full

advantage of the medium is a more challenging task. You must pay attention to simple things like data organization and window size and position on the disc. If a CD-ROM is not well organized, your user might as well be dealing with 825 individual floppy disks. Ideally, you want to present a consistent interface and make it as visually appealing as possible to the user.

DISTRIBUTING LARGE PRODUCTS

The simplest way to use CD-ROM is as a distribution medium for relatively large and complex software products. Development system environments and other comprehensive products like them are perfect candidates for CD-ROM distribution. With the increased disk space, you can include several different preconfigured versions that a user need only copy to a hard disk without going through a complicated installation procedure.

If your product requires a specific version of Apple's system software, you can license the software from Apple Software Licensing (20525 Mariani Avenue, M/S 38-I, Cupertino, CA 95014, 408/974-4667) and include it on the CD-ROM to make sure your users have everything they need to run your product when they acquire it.

For Apple, products like MPW®, MacApp®, and System Software 7.0 are great candidates for CD-ROM distribution, but the real win with CD-ROM technology is to create products that you may not have even thought possible prior to the availability of this medium.

CREATING NEW OR ENHANCED VERSIONS OF PRODUCTS

CD-ROM gives developers the opportunity to produce new or enhanced versions of their products that just would not be feasible with floppy disk-based distribution. These enhancements can include expanded help systems (since basic on-line help should already be in every good application), full-blown tutorials that use graphics, sound, and animation to provide users with a rich learning environment, advanced technical or support information, and any other supplemental components that add value to the original product. Developers can even include past versions of the product to facilitate upgrade compatibility and file exchange.

Highlighted Data's new electronic version of the *Merriam-Webster's Ninth New Collegiate Dictionary* would not have been possible without CD-ROM technology. It features digitally recorded pronunciations of more than 160,000 entry words, as well as the full text and graphics of the print edition.

The *Manhole* CD-ROM by Mediagenic, the industry's first CD-ROM entertainment product for the Macintosh, is a great example of an enhanced product. It was developed from the popular HyperCard-based fantasy adventure. On the CD-ROM, enhanced graphics and animation combine with original music to bring the characters of this adventure to life.

DISTRIBUTING APPLICATION ENVIRONMENTS

Some developers are taking advantage of CD-ROM to produce and distribute "complete" application environments for business and other uses. The concept behind an application environment is simple: combine several individual products that complement each other, in a way that appears seamless to the user. The resulting savings on packaging and distribution costs can be passed along to users.

With the data encryption techniques already available for CD-ROM, you can include your entire line of software on a single CD-ROM and distribute it with different decryption keys. When a user decides to add another component, he or she purchases the key to decrypt another application from the same CD-ROM.

To date, the best-known example of an application environment is the Microsoft Office. Microsoft bundles their four most popular business applications—Word, Excel, PowerPoint, and Mail—on a CD-ROM with on-line documentation, HyperCard-based navigational tools, popular templates and clip art, and several third-party applications that add even more value to the suite of applications.

ENABLING COLLABORATIVE PRODUCTS

A related area with a great deal of potential for CD-ROM is cooperative or collaborative works put together by multiple developers or a publisher and several developers. The idea is for a publisher or group of developers to combine several related applications, texts, graphic images, sounds, or animations into a user environment that the individual developers would not have considered producing, or could not have produced, individually. This approach can work especially well for smaller developers who cannot lavish money on product packaging and promotion. Collaborative works also have a great deal of potential for users, since they get the combined efforts of several developers who have worked together on a single product.

DISTRIBUTING INFORMATION PRODUCTS

Information products are natural candidates for distribution on CD-ROM, as this technology very easily lends itself to storing immense libraries of information that can be indexed, cross-referenced, and searched electronically. The data in these libraries can be text, sound, graphics, animation, or anything else you happen to fancy.

Apple is seriously investigating producing a CD-ROM of its entire technical library for developers, including the Technical Notes series, *Inside Macintosh*, and MacApp documentation. The *Developer Essentials* disc is just the tip of the proverbial iceberg; see Corey Vian's sidebar for details. OCLC (On-Line Computer Library Corporation) publishes an entire line of CD-ROMs for use in libraries around the world. The *MacroMind CD-ROM* by MacroMind, Inc., includes over 100 megabytes of VideoWorks and MacroMind Director animations, guided tours, and clip art. These are all first-generation examples of the use of CD-ROM to distribute information products.

Developers have just begun to scratch the surface of what is possible when a local CD-ROM archive is combined with up-to-date on-line data, a combination that offers users both the storage advantage of CD-ROM and the timeliness that only an on-line system can offer. The normally high cost and slow speed of on-line system searching is negated by the presence of the local CD-ROM archives. The "AppleLink Offline" stack produced by DTS, currently distributed to Apple Partners and Associates, is a good prototype of this combination.

Beyond these archives, or knowledgebases, there is also great potential for combination CD-ROM and on-line applications where large amounts of data could be stored locally on CD-ROM and the code for interacting with other on-line users and accessing the data on the CD-ROM could be stored on an on-line system.

INSIDE AN ELECTRONIC d e v e l o p

by Corey Vian, Apple Developer Press

Electronic publishing is relatively unexplored territory and so invites both innovation and disaster. When we decided to take d e v e l o p electronic, we knew we would see a little of each. Initially we thought that the electronic version should precisely match the printed version. Our reasons for wanting to maintain parallelism between the printed and electronic version had to do with spatial orientation and translation efficiency.

Some of us remember the location of the content we have read. We remember that a reference was on the lower-right portion of a left-hand page near the end of the magazine, for example. If we read it first in the printed version and later want to find it in the electronic version, this spatial parallelism makes switching media much easier.

The idea of taking a printed magazine—of any format—and making a page-by-page translation into electronic form is attractive in that it generalizes the process of translation and eliminates some production time by reusing work already done on the printed version. We imagined that any publication could be translated into electronic form using a standard, nearly automatic conversion process.

How We've Done It. We created the first issue of d e v e l o p by printing PageMaker documents to glue files, converting the glue files to PICT files, and pasting them into SuperCard. We ran into numerous problems because the PICT files were being interpreted differently at each step. Some PICT interpreters make a single object out of an entire line of text, while others make individual objects out of each word in an attempt to preserve spacing when fonts are changed. If the original font used in the PICT is not available, a substitute font is used. We copied the text for each page into an invisible field in its corresponding card so that when you searched for text, you were taken to the appropriate card. The drawback was that we could not highlight the text once found, since it was embedded in the PICT.

We explored the possibility of writing XCMDs that would actually search a PICT file and highlight the found string, which seemed doable but only addressed a small part of the problem. Neither the process nor product for the first issue were turning out quite as we had envisioned. It was too slow, it displayed its pages without respect for the size of the screen it was being viewed on, PICT files were misinterpreted in the translation process, and the pages were difficult to read overall. After all, they were designed to be printed at

1200 dpi. Basically, the publication showed no respect for the medium it was being translated into. So we decided to take a step back and reevaluate our approach. With deadlines looming, we agreed on a slightly improved interim remedy and began the process of designing a more elegant, long-term solution.

For the second issue, we scrapped the full-page PICT approach. Instead, we copied the text into SuperCard fields, copied the pictures individually, restyled the text, and then placed the pictures to approximate the layout of the journal. This was surprisingly easy and took only a day or two. It also gave us more readable text and the ability to search and highlight the actual text you were reading. We still included full-page PICTs elsewhere in the application, used for printing.

For the third issue, we have reformatted the previous issues in this same way and included them all in one file so that all articles are directly accessible from a single application. We will handle subsequent issues this way until we have worked out a design that better addresses key problems inherent in electronic publishing.

Where We're Headed. As with life, one of the most basic problems associated with an electronic publication is the unknown. When we distribute a printed publication, the content, presentation hardware, and interface are all bundled together under our control. With electronic publications, there are many variables over which we have little or no control—primarily screen size, resolution, and color limitations, as well as processing speed. This places severe restrictions on the design process.

Should we design for the lowest common denominator and flush innovation down the toilet? Should we design for the average platform and irritate many subscribers? Or can we design a presentation that will work well in the various environments and still let us exploit the extraordinary capabilities of the computer? After all, we could simply put all the articles up on AppleLink and call it a day. The challenge is to develop a standardizable framework that will work for users and have a minimal impact on creative options.

Most media have arrived at some consistent format. Some constraints are put in place to provide context and a familiar, and therefore comfortable, entry into the content, like the frame around a painting, the form of a musical fugue, or the size of a magazine as compared, say, to a dictionary. Yet within those constraints, substantial creativity is possible. A balance can be struck between the need for consistency and the need for creativity. We are struggling to find this balance in the realm of electronic publications, but the rich and rapidly changing nature of the medium resists our efforts.

What We've Learned. We've learned that page-for-page translation is probably not a good idea. Formatting that is left- or right-handed makes no sense on a screen that displays a single page at a time—if that. Using scroll bars to push an image around in order to see the whole thing is at best a work-around. Magazine print is too small to be viewed at 72 dpi without enlarging. A solution that takes full advantage of, yet respects the limitations of, the electronic environment will feel far more natural than trying to dress a print man in a cathode-ray suit.

It is a mistake to think of the printed version of d e v e l o p as the basic product to be recast in an electronic form. d e v e l o p, or any other publication, is not essentially a collection of printed pages. Rather, it is a specific domain of information, knowledge if you like, that can be conveyed by a variety of media. Effective use of the electronic environment demands a fresh look at what publishing can be.

ENABLING INTERACTIVE MEDIA PRODUCTS

Because CD-ROM stores all of its information in digital format, it's an excellent medium for combining text, graphic images, sound, animation, and any other type of data for interactive media. The only real limitation is retrieval speed from the disc, and at 150K per second, CD-ROM is fast enough for just about anything but uncompressed, full-motion color video.

To date the largest uses of interactive media on CD-ROM have involved HyperCard, which has proven worthy of the task. With HyperCard 2.0, it is even easier to author stacks that combine sound, color graphics, animation, and related data for use on CD-ROM; and the new features of HyperCard 2.0 stacks are no longer limited by disk space.

Audio Notes #1: "The Magic Flute," produced by Warner New Media, is an excellent example of interactive media on CD-ROM. This product, the first in a Music Discovery series, fills three CD-ROM discs with both HyperCard stacks and CD-ROM audio content on Mozart's *The Magic Flute.* Roger Englander, the television producer of Bernstein's Young People's Concerts, provides music commentary to go with the original German libretto and the English translation. In addition, this CD-ROM set includes newly recorded sidebars about the music, the symbolism, and opera forms.

Ludwig Van Beethoven, Symphony No. 9, another example of interactive media, is a part of the Voyager CD Companion Series from the Voyager Company. This product by Robert Winter is a commercially produced audio CD-ROM with accompanying floppy disks of data and HyperCard stacks. These stacks not only teach the user about Beethoven and the symphony itself, but also about music theory in general.

ENABLING USER INTERACTION IN A LEARNING ENVIRONMENT

Imagine a CD-ROM about film that enables you to direct your own scenes with different camera angles and effects; or a CD-ROM that guides you through the city of your choice as it looked or will look at a specified time period; or a CD-ROM that gives you the complete works of French mathematician and philosopher Blaise Pascal in both French and English (or any other language) with pictures of his surroundings at the time of his writings; timelines of the major events during his life; excerpts from popular music, literature, and architecture of the period; and critiques and comments from his contemporaries.

Although it may have the greatest potential for both instructional and general use, the area of "user views" is one of the least developed at this time. The idea of a user view is that the developer provides the user with an environment in which he or she is guided by interactive tutorials and interpretive simulations through the data, and then enables the user to build his or her own "views" of the same data. At any time

the user can switch between the different types of available data and the selected view. CD-ROM is the ideal medium for this type of thing, since it can contain texts, graphic images, sounds, statistics, timelines, and animations.

WHAT NOW?

This article has given you a few ideas to get you started thinking about how you can best benefit from CD-ROM technology, and you can probably think of many more. The point is that CD-ROM opens a lot of doors and removes a lot of assumed barriers to product development. This technology offers you a rare opportunity to let your imagination run wild; however, what you choose to do with it is up to you.

For More Information

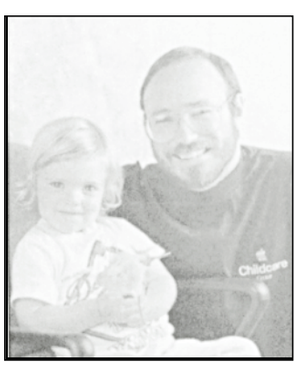
Sources used in this article were *CD-ROM and the Macintosh Computer* by Brian Bechtel (on the accompanying *Developer Essentials* disc), and Gregg Williams's article "The Big Advantages of CD-ROM" in *Apple Direct,* December 1989.

For a good overview of CD-ROM, read the *AppleCD SC Developers Guide,* available from APDA. In addition, there is a magazine called *CD-ROM EndUser,* available from DDRI, 510 North Washington Street, Suite 401, Falls Church, Virginia 22046.

Thanks to Our Technical Reviewers:
Andy Poggio, Llew Roberts

THE INS AND OUTS OF ISO 9660 AND HIGH SIERRA

Any CD-ROM can be read at the bit level by any CD-ROM player, thanks to the existence of standards for the physical format of such discs. Having this physical format in common is nice, but it's not enough. We also need to be able to find specific files on a CD, no matter which operating system we are using; we need a standard file system format. High Sierra and its international equivalent ISO 9660 are standards that define a file system usable under a variety of operating systems. This article explores these standards and their implementation on the Macintosh, and discusses a simple program you'll find on the accompanying Developer Essentials disc to convert Macintosh files to ISO 9660 format.



BRIAN BECHTEL WITH HIS DAUGHTER MEG

A file system organizes data logically on a CD. Different operating systems use different file systems to organize data, and thus a CD formatted with a native file system can only be read by one particular operating system. To overcome this obvious limit to the usefulness of CD-ROM as a storage and distribution medium, the industry has established standards for a file system that can be used under a variety of operating systems. The ISO 9660 standard and its predecessor High Sierra define a file system carefully attuned to CD-ROM characteristics.

In particular, because CDs have a relatively slow seek time and a high capacity, these standards make trade-offs that reduce the number of seeks needed to read a file, at the expense of space efficiency. And because CDs are read-only, concerns like space allocation, file deletion, and the like are not addressed in the standards. The standards apply only to the data track of a CD-ROM, not to audio tracks; and they do not apply to any media other than CD-ROM, such as erasable-optical drives.

BRIAN BECHTEL works in the Advanced Technology Group, where he applies to his everyday life Wernher Von Braun's slogan, "Research is what I do when I don't know what I'm doing." His title of Witzelsuchter is derived from an obscure medical condition (usually caused by brain lesions) in which the patient takes an intense interest in telling long, pointless stories and jokes. People who know him say this title is appropriate. Brian claims the lesions resulted from nine months of studying the ISO 9660 and High Sierra standards documents. He also wrote the *HyperCard CD Audio Toolkit*. He graduated from Occidental College with an A.B. in math. Meg, his daughter, attends the Apple Child Care Center. His favorite food is chocolate,

The standards do not favor any particular computer architecture. All significant multibyte numbers are recorded twice, once with the most significant byte first (msb order, used by Intel processors such as those in MS-DOS compatible computers) and once with the least significant byte first (lsb order, used by Motorola microprocessors such as those in the Macintosh). This enables easy implementation under a variety of operating systems, such as the Macintosh operating system, Apple II ProDOS 16 or GS/OS, MS-DOS, VMS, and the UNIX operating system. Let's look now at how the two standards developed.

ISO 9660 AND HIGH SIERRA: SOME HISTORY

A group of industry representatives met at Del Webb's High Sierra Hotel and Casino at Lake Tahoe, Nevada, in late 1985 to see if companies could cooperate in developing a common file system format for CD-ROM. The result of this series of meetings was the High Sierra format. This format is fully specified by the May 28, 1986 *Working Paper for Information Processing—Volume and File Structure of Compact Read-Only Optical Discs for Information Interchange*. For obvious reasons, this is known as the High Sierra paper.

The world at large then wanted to adopt an equivalent standard. The International Organization for Standardization pushed High Sierra through its standardization process, resulting in the international standard known as ISO 9660. (The organization is called the International Organization for Standardization, but the standard is *ISO 9660*.) This standard is described in the paper *ISO 9660—Volume and File Structure of CD-ROM for Information Interchange*, known in the CD-ROM trade as the ISO standard.

Apple's Macintosh operating system and GS/OS, plus Microsoft's operating system MS-DOS, support both the ISO 9660 standard and the older High Sierra format.

ISO 9660 is the wave of the future—many existing CD-ROMs use the High Sierra format, but everyone is changing over to the ISO 9660 standard, and most if not all future discs will be in ISO 9660 format rather than High Sierra format. In the meantime, because "ISO 9660" doesn't roll off the tongue quite as nicely as "High Sierra," many people in the industry say "High Sierra" when they really mean "ISO 9660" or "whatever that damn format is that my CD-ROM is supposed to be in." In this article, I do not use the terms interchangeably, but explicitly state which format I'm referring to. But for practical purposes, what I say about one format also applies to the other, with the exceptions I note.

HOW THE FORMATS ARE IMPLEMENTED ON THE MACINTOSH

The Macintosh supports both ISO 9660 and High Sierra through the use of a feature in the Macintosh file system called the external file system hook. This is a

as is his favorite color. He says he plays lousy acoustic guitar and roots for the LA Dodgers. His identical twin, Bradley, manages technical support at some other Silicon Valley company.

low-memory global that contains a pointer to an external file system handler to which multiple handlers are daisy-chained. To support ISO 9660 and High Sierra, Apple has written a new set of routines, contained in a file called Foreign File Access. This file, combined with the files High Sierra File Access and ISO 9660 File Access, provides complete support for the standard formats.

Because the ISO 9660 and High Sierra formats are supported via Foreign File Access instead of software that's part of a device driver, you can use any media to create a standard-format volume. In actual use, ISO 9660 and High Sierra only make sense on a CD-ROM; but you can create a test volume using any floppy or hard disk.

A LOOK AT THE FORMATS

The ISO 9660 standard and the older High Sierra format define each CD-ROM as a volume. Volumes can contain standard file structures, coded character set file structures for character encoding other than ASCII, or boot records. Boot records can contain either data or program code that may be needed by systems or applications. ISO 9660 and High Sierra specify

- how to describe an arbitrary location on the volume—the logical format of the volume;
- how to format and what to include in the descriptive information contained by each volume about itself—the volume descriptors;
- how to format and what to include in the path table, which is an easy way to get to any directory on the volume;
- how to format and what to include in the file directories and the directory records, which contain basic information about the files on the volume such as the filename, file size, file location, and so forth.

The discussion that follows is a reasonably technical description of the standards in each of these areas; it is *not* the definitive description. For the one true, proper definition of the standards, read the original specifications.

THE LOGICAL FORMAT

CD-ROMs are laid out in 2048-byte physical sectors. This physical layout is defined in a standard published by Philips and Sony known as the Yellow Book, and is independent of the type of volume formatting used. Under ISO 9660 and High Sierra, the CD is also laid out in 2048-byte logical sectors. Both formats also have the concept of a logical block, which is the smallest chunk of file data. A logical block can be 512, 1024, or 2048 bytes. In general, file access information is laid out in sector-sized units, while actual file data is laid out in block-sized units. On most CDs, the block size is the same as the sector size at 2048 bytes, so this distinction isn't important. Figure 1 shows the layout of a volume in ISO 9660 or High Sierra format.

If you're really interested in the standards, you should get copies of the full specifications. You can get the May 28, 1986, High Sierra specification from

National Institute of Standards and Technology
Administration 101
Library E-106
Gaithersburg, MD 20899

You can get the ISO 9660 specification from any of the following:

American National Standards Institute
1430 Broadway
New York, NY 10018
Sales Department: 212/642-4900

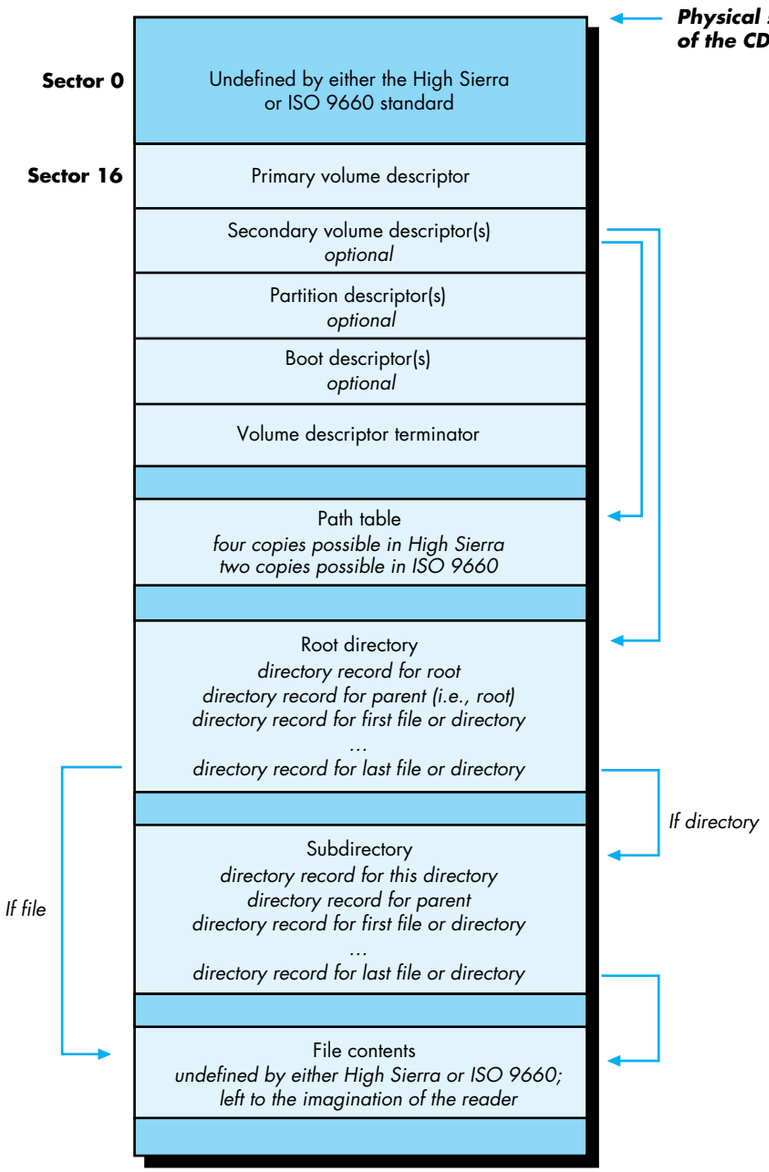


Figure 1
A Volume in ISO 9660 or High Sierra Format

ECMA Headquarters
Rue du Rhone 114
CH-1204
Geneva, Switzerland

Global Engineering Documents
800/854-7175 or 714/261-1455

THE VOLUME DESCRIPTORS

Information about the volume itself is contained in an array of 2048-byte entries, beginning at logical sector 16 on the disc, as shown in Figure 1. These are the volume descriptors. There are five types of volume descriptors: the primary volume descriptor, the secondary volume descriptor, the boot descriptor, the partition descriptor, and the volume descriptor terminator. Every volume descriptor is 2048 bytes long (one sector). The first descriptor in the array is always a primary volume descriptor, and the last descriptor always a volume descriptor terminator. The other three volume descriptor types are optional. The boot descriptor and the partition descriptor aren't supported by the Macintosh, because the Macintosh boot code looks at the beginning of the disk for boot tracks, not at sector 16.

Each volume has one and only one *primary volume descriptor*. This descriptor consists of the volume name, some publishing information, and offsets to the path table and root directory. The primary volume descriptor also contains a copy of the root directory entry (to minimize the number of seeks necessary to find out information about a disc). In the directory structure pointed to by the primary volume descriptor, filenames can consist of the uppercase characters A through Z, the underscore, and the digits 0 through 9. This is a subset of ISO 646, an international character representation standard roughly equivalent to ASCII. You will see a sample primary volume descriptor later in this article in the section entitled "A Simple Formatting Program: ISO 9660 Floppy Builder."

A volume can have zero or more *secondary volume descriptors*. The purpose of the secondary volume descriptor is to enable you to press a CD-ROM that can display the directories in a nonroman character set, such as Japanese Kanji, Hebrew, or Arabic. In the directory structure pointed to by the secondary volume descriptor, the characters used to represent filenames are not restricted to ISO 646. This directory structure is separate from but parallel to the directory structure pointed to by the primary volume descriptor. The secondary volume descriptor contains the same information as the primary volume descriptor—although in a different alphabet—in all but two fields. The **`volumeFlag`** field is used to indicate whether a non-ISO-standard alphabet is being used. The **`escapeSequences`** field contains characters that define which alphabet is being used.

The files ISO 9660 File Access and High Sierra File Access each contain a resource used to determine if the Macintosh should use a secondary volume descriptor. The NRVD resource contains a word for the **`volumeFlags`** field, followed by 32 bytes for the **`escapeSequences`** field. If a secondary volume descriptor exists, and if the volume flags and escape sequences match those in the NRVD resource, then the secondary volume descriptor is used instead of the primary volume descriptor.

The *boot descriptor* was designed to allow the creator of a CD-ROM to include system information for booting from that CD-ROM. This descriptor is not

For the five people out there who really care: Apple's High Sierra and ISO access software supports level-2 interchange, according to section 10.2 of the ISO 9660 specification. This means it supports interleaved files, but not multivolume sets.

supported on the Macintosh, since the Macintosh operating system looks for boot information at the beginning of the disk, in the area undefined by ISO 9660 and High Sierra. The *partition descriptor* is also unsupported on the Macintosh.

The *volume descriptor terminator* is a simple structure that serves to indicate the end of the volume descriptor array. Each volume contains one, and only one, volume descriptor terminator.

THE PATH TABLE

The *path table* describes the directory hierarchy in a compact form, containing entries for each of the volume's directories. Its purpose is to minimize the number of seeks necessary to get to a file's directory information. The Macintosh caches the path table in memory, enabling access to any directory with only a single seek.

ISO 9660 allows up to two identical copies of the path table to be stored on the disc, while High Sierra allows up to four copies. This is useful to operating systems that do not cache the path table in memory. In this case, copies of the path table can be stored at regular intervals on the disc—say a quarter of the way in and again three-quarters of the way in—to decrease the seek time necessary for the optical read head to find one of the copies.

The path table for a simple formatting program is shown later in this article.

DIRECTORIES

Directories are stored in a hierarchical tree. Each volume has a root directory, the parent to all other directories on the volume. Subdirectories can be nested up to eight levels deep (the root plus seven levels).

Directory records are the basic unit of information kept about each file. Each directory record contains the offset from the beginning of the disc to the file itself, the size of the file, date and time information for creation and modification, file attribute flags, information useful for interleaved files, and the filename (preceded by a length byte). There is also an optional extension field, used by the Macintosh and Apple II operating systems to store additional information not defined by the High Sierra and ISO 9660 formats but necessary to the operating system. A directory record for a simple formatting program is shown later in this article.

Additional file information necessary for multiuser operating systems such as the UNIX operating system or VMS is retained in a separate field known as the extended attribute record. Extended attribute records are recognized by the Macintosh, but they are ignored since they contain information that is irrelevant to it.

A file identifier consists of a filename, a period, a file extension, a semicolon, and a file version number. File identifiers can use the uppercase English alphabet, numbers, and the underscore character (_), and can be up to 31 characters long.

Either the filename or file extension can be missing, but not both; if the extension is missing, the period must still precede the semicolon; and the version number must exist. This means that valid file identifiers look like THIS_FILE.EXISTS;1 or .ONLYEXTENSION;1 but that file identifiers like NO_PERIOD;1 or NO_VERSION are invalid. Both standards define a level-1 conformance, designed for compatibility with MS-DOS, that restricts filenames to eight characters, a period, three characters, a semicolon, and a version number.

There are two types of files: regular files and associated files. A regular file without an associated file is simply a stream of bytes, like the files used in an operating system such as the UNIX® operating system or MS-DOS. An associated file is a file with the same name as a regular file, and with the associated file attribute bit set in the directory record. This scheme accommodates the data and resource forks of a Macintosh file, as we'll discuss later.

HOW THE FORMATS DIFFER

The differences between ISO 9660 and High Sierra are slight, and mostly of interest to programmers. They are as follows:

- The primary and secondary volume descriptors differ in the type and number of fields they accommodate.
- In ISO 9660, a bibliographic preparer field was added to the primary and secondary volume descriptors.
- Up to four copies of the path table are allowed in High Sierra, but only two copies in ISO 9660.
- Two fields changed position in the directory records in ISO 9660.
- All date/time fields have an extra byte in ISO 9660, used to describe the 15-minute offset from Universal Standard Time (GMT or UTC).
- The order of directory records is slightly different in ISO 9660. In High Sierra, the associated file comes after the regular file with which it is associated; in ISO 9660, the associated file comes first.

HOW MACINTOSH FILES ARE STORED IN THE FORMATS

The Macintosh uses a file system called the Hierarchical File System (HFS). As its name implies, it is hierarchical in structure, like that specified by ISO 9660 and High Sierra; it supports subdirectories, called folders, where files can be logically grouped together. HFS corresponds reasonably well to the ISO 9660 and High Sierra formats, with some limitations. Let's look at specific parts of the information required by the Finder and see how the ISO 9660 and High Sierra support handles these issues.

FILE FORKS

Every file in HFS has two forks: a resource fork and a data fork. The resource fork of an application file contains the resources used by the application (for example, the

bit image for an icon or the title and commands for a menu) plus the application code. The data fork can contain anything an application wants to store there. Similarly, a document file contains the document's resources in its resource fork and the document's data in its data fork. In ISO 9660 and High Sierra format, the data fork of a Macintosh file is stored as a regular file, and the resource fork is stored as an associated file.

A Macintosh application's data fork may be empty. How this should be handled is not stated clearly in either the ISO 9660 or the High Sierra specification; however, in both cases, an associated file is defined to exist only in conjunction with a regular file of the same name. If the regular file (corresponding to the data fork) is missing, the Macintosh operating system handles the case correctly; however, MS-DOS won't show the file, because the MS-DOS CD-ROM extensions ignore files with the associated bit set. This is because all files in MS-DOS are regular files.

FILE IDENTIFIERS

Like ISO 9660 and High Sierra file identifiers, HFS filenames can have a maximum of 31 characters. HFS filenames differ from valid ISO 9660 and High Sierra file identifiers in the following ways:

- HFS does not distinguish between uppercase and lowercase letters; the names "forecast," "Forecast," and "FoReCaSt" all refer to the same file.
- HFS allows any character to be used in a filename except the colon (:). This means that filenames such as "My payroll file" or "Åéîøü" are perfectly acceptable on the Macintosh.
- In HFS there is no concept of a filename extension. File types are stored as part of the Finder information.

These differences mean that many HFS filenames are illegal in ISO 9660 or High Sierra format. This may cause problems in an application that depends on hard-coded filenames. For example, Hypercard requires that the home stack be named HOME, but this is illegal in ISO 9660 and High Sierra. The legal ISO 9660 or High Sierra name is HOME.;1, which won't be found by Hypercard. Some versions of Videoworks depend upon sounds being in a file named Sounds. The only solution is to have the user copy such files over to an HFS volume and rename them.

FILE TYPE AND CREATOR

To establish the proper interface with the Finder, when a Macintosh application creates a file it sets the file's creator and file type. Normally it sets the creator to its signature, which is a unique four-letter sequence by which the Finder can identify it, such as MACA for MacWrite, XCEL for Excel, and FNDR for Finder. It sets the file type to a four-character sequence that identifies files of that type, such as TEXT for plain text or documents of unknown type, APPL for applications, and WORD for MacWrite documents. When the user asks the Finder to open or print the file, the

Finder starts up the application whose signature is the file's creator and passes the file type to the application, along with the filename and other identifying information. This information about each file is not defined in either High Sierra or the ISO 9660 standard. To preserve this file-specific information, Apple has defined a legitimate extension to ISO 9660 (which also applies to High Sierra), documented in *CD-ROM and the Macintosh Computer*, included on the *Developer Essentials* disc. The extension specifies how to use the optional **`SystemUse`** field present in each ISO 9660 directory record to accommodate the file type and file creator.

If a CD-ROM has been pressed in ISO 9660 or High Sierra format without the Apple extension, all files on the disc are considered to be of type TEXT and creator hscd. TEXT is a generic type that can be read successfully by many Macintosh applications; hscd is a creator registered with Developer Technical Support that does not correspond to any application or utility. If the CD-ROM has been pressed with the Apple extension, then files on the disc can have any arbitrary type and creator.

FINDER FLAGS

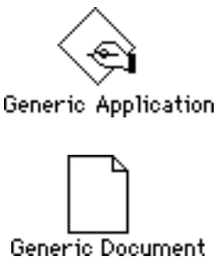
The Finder flags are defined in Technical Note #40, Finder Flags. Only the invisible bit has an analogy in the ISO 9660 and High Sierra formats, but with the Apple extension to ISO 9660, the **`SystemUse`** field in the directory record accommodates Finder flags. If the CD-ROM has been pressed with the Apple extension, only bits 5 (always switch launch), 12 (system file), 13 (bundle bit), and 15 (locked) can be used. All other bits are either ignored or set due to internal workings of the file system translator. Flags indicating that a file is on the desktop or in the trash are not supported; all files are assumed to be in their folders.

DESKTOP INFORMATION

The Finder also requires some information describing how files on the desktop are to be viewed, the icon to display for a specific file, the position of folders and file icons on the desktop, and the default scroll position when the user opens a folder. This information is contained in the **`FInfo`**, **`FXInfo`**, **`DInfo`**, and **`DXInfo`** structures documented on pages IV-104 through IV-106 of *Inside Macintosh*. File or folder comments are kept in the Desktop file. None of this information can be specified when pressing a CD-ROM in ISO 9660 or High Sierra format. Some of it is computed by the ISO 9660 File Access or High Sierra File Access software, however.

Due to some deficiencies in the original design of the Finder, the correct icon cannot be displayed for a file on an ISO 9660 or a High Sierra disc. This is because the Finder does not actually ask for the icon of a file; rather, it assumes the existence of a *desktop database* that contains these mappings, and makes a special call, giving only the file creator and type. The software to provide this information was designed to be very HFS-specific. Currently, even if the icon bitmap for a file on an ISO 9660 or a High Sierra disc is defined in the Apple extension, it is not used by the Finder.

or a High Sierra disc is defined in the Apple extension, it is not used by the Finder.



Consequently, all files on a High Sierra or an ISO 9660 disc display a generic icon. If such a file is copied to a hard disk, the correct icon is then displayed on the desktop. If the user double-clicks on a generic application icon, the application opens correctly. If the user double-clicks on a generic document icon, and the associated application exists only on CD-ROM and not in the current directory, the application will not be found; if the application exists on an HFS volume (because the user has copied it there), it will be found.

Under HFS, the Finder keeps track of the position of a file icon on the desktop or in a folder by using a special field; under High Sierra and ISO 9660, an icon s position is computed when the folder is opened, and cannot be changed. File and folder comments are not supported under the ISO 9660 and High Sierra formats. The view is always assumed to be View by Icon and the scroll position is always assumed to be at the top of the folder; these items are hard-coded in the file system translators.

SUMMARY

As a developer, you don t have to worry about files on an ISO 9660 or a High Sierra CD-ROM looking different to your application. You may have to worry about filenames, if you have hard-coded a particular filename into your application (which is always a bad idea anyway.) Except for the icons not showing up properly (a major exception), your users don t really see a difference between ISO 9660, High Sierra, and HFS-format CD-ROMs. Names are reported back to the Finder exactly as found on the High Sierra or ISO 9660 volume; they are not altered in any way, except that they are truncated at 31 characters if they started out longer.

STRANGE BEHAVIOR IN ISO 9660 AND HIGH SIERRA SUPPORT

Version 1 of ISO 9660 File Access and High Sierra File Access had a misfeature that slowed down volume mounting times on CD-ROMs with a large number of files. Because neither the ISO 9660 nor the High Sierra format contains a count of the total number of files on a volume, the access software was iterating over the volume to find this number to stuff into the volume control block. This could make a CD-ROM with 10,000 files on it take up to 20 minutes to mount.

It turns out that the volume control block field that was being set is used in only one place in the Macintosh operating system: the file count of the **`GetInfo`** of the volume. Version 2 of High Sierra File Access and ISO 9660 File Access fixes this problem by setting the appropriate field in the volume control block to 0. A special hard-coded comment has been added to the volume s **`GetInfo`** box that says either The number of files shown is incorrect due to limitations of the High Sierra format or The number of files shown is incorrect due to limitations of the ISO

SO HOW DO I PRESS AN ISO 9660 CD-ROM?

CD-ROMs are actually pressed from an image of a disk. To press a CD-ROM in ISO 9660 or High Sierra format, you need some premastering software that creates a disk in the appropriate format. You can either hire a CD-ROM pressing plant to convert your files to the ISO 9660 format, or you can purchase a system to do it yourself, or you can write your own ISO 9660 formatting software.

COMPANIES TO CONTACT FOR CD-ROM PRODUCTION

Here's a list of pressing plants that can convert your files to the ISO 9660 format:

3M Optical Recording
Building 223-5S-01
3M Center
St. Paul, MN 55144
612/736-3274
Mark Arps/Dick Tendill
AppleLink: D2462

DADC
1800 N. Fruitridge Ave.
Terre Haute, IN 47804
812/462-8100
Linda Watson/Kozo Arai
AppleLink: D2125

Nimbus Information Systems
SR 629, Guildford Farm
Ruckersville, VA 22968
800/782-0778
Larry Boden

Denon America
222 New Road
Parsippany, NJ 07054
201/575-2532
Nob Tokutake/Ben Garcia

Disctronics
1120 Cosby Way
Anaheim, CA 92806
714/630-6700
Wan Seegmiller

Philips Dupont Optical
1409 Foulk Road
Suite 200
Wilmington, DE 19803
800/433-3475
Jill Jones
AppleLink: D2173

If you want to buy your own premastering system, you can contact one of the following:

Meridian Data, Inc.
5615 Scotts Valley Drive
Scotts Valley, CA 95066
408/438-3100
Dean Quarnstrom

Optical Media Int'l.
485 Alberto Way
Los Gatos, CA 95032
408/395-4332
Applelink: D1490

(The legal beagles want me to tell you that these company names are here for your information, but Apple Computer, Inc., does not recommend or endorse any particular company listed.)

If you want to write your own software, you'll find a simple example program on the *Developer Essentials* disc to get you started. The program is called ISO 9660 Floppy Builder and is written in Think C. It builds disks conforming to the ISO 9660 standard.

A SIMPLE FORMATTING PROGRAM: ISO 9660 FLOPPY BUILDER

ISO 9660 Floppy Builder has a number of features, listed below. The bracketed number after each feature indicates the section in the formal ISO 9660 document referred to earlier that describes this feature. You should read that section of the ISO 9660 document for more detail about each feature.

- Assumes that the logical sector size is 2048. [6.1.2]
- Assumes that the logical block size is 2048. [6.2.2]
- Writes a primary volume descriptor. [8.4]
- Writes a volume descriptor terminator. [8.3]
- Supports the Apple extensions to ISO 9660.
- Enables the user to specify a volume name. The volume name is automatically converted to the proper character subset. [8.4.6]
- Enables the user to choose, via a standard file dialog, files to be added to the ISO 9660 disk. All files are currently put in the root directory.
- Copies both the resource fork and the data fork of a file to the disk. The resource fork is stored as the associated file.

ISO 9660 Floppy Builder is a demonstration program; it doesn't do many of the difficult parts of building a disk in ISO 9660 format. Specifically, it doesn't support: subdirectories (folders), keeping the files in a directory in alphabetical order, a main directory whose total size exceeds one block of 2048 bytes, a block size other than 2048 bytes, secondary volume descriptors (used to implement non-ASCII alphabets), or more than one logical sector of directory records.

TO USE THE PROGRAM

To use the program, start with a formatted blank floppy. We will unmount and format the disk as part of the process of making it into an ISO 9660 format disk. All data will be lost from the floppy inserted.

Select "Specify Files for Root…" to put files into the root directory. You'll be asked for the names of the files to be copied over via a standard file dialog. When you've finished selecting filenames, click the Cancel button.

A CLOSER LOOK AT THE CODE

Let's look at the C structures we'll use to implement ISO 9660. We need three basic data structures: the primary volume descriptor, the path table, and the directory record. A primary volume descriptor has the basic data for the entire volume. It looks like this in C:

```
typedef unsigned char Byte;
typedef unsigned short Word;
typedef unsigned long Long;

typedef struct
{
   Byte  VDType;              /* Must be 1 for primary volume descriptor. */
   char  VSStdId[5];          /* Must be "CD001". */
   Byte  VSStdVersion;        /* Must be 1. */
   Byte  volumeFlags;         /* 0 in primary volume descriptor. */
   char  systemIdentifier[32];/* What system this CD-ROM is meant for. */
   char  volumeIdentifier[32];/* The volume name. */
   char  Reserved2[8];        /* Must be 0's. */
   Long  lsbVolumeSpaceSize;  /* Volume size, least-significant-byte order. */
   Long  msbVolumeSpaceSize;  /* Volume size, most-significant-byte order. */
   char  escapeSequences[32]; /* 0's in primary volume descriptor */
   Word  lsbVolumeSetSize;    /* Number of volumes in volume set (must be 1). */
   Word  msbVolumeSetSize;
   Word  lsbVolumeSetSequenceNumber;/* Which volume in volume set (not used). */
   Word  msbVolumeSetSequenceNumber;
   Word  lsbLogicalBlockSize; /* We'll assume 2048 for block size. */
   Word  msbLogicalBlockSize;
   Long  lsbPathTableSize;    /* How many bytes in path table. */
   Long  msbPathTableSize;
   Long  lsbPathTable1;       /* Mandatory occurrence. */
   Long  lsbPathTable2;       /* Optional occurrence. */
   Long  msbPathTable1;       /* Mandatory occurrence. */
   Long  msbPathTable2;       /* Optional occurrence. */
   char  rootDirectoryRecord[34];   /* Duplicate root directory entry. */
   char  volumeSetIdentifier[128];  /* Various copyright and control fields */
                                    /* follow. */
   char  publisherIdentifier[128];
   char  dataPreparerIdentifier[128];
   char  applicationIdentifier[128];
   char  copyrightFileIdentifier[37];
   char  abstractFileIdentifier[37];
   char  bibliographicFileIdentifier[37];
   char  volumeCreation[17];
   char  volumeModification[17];
   char  volumeExpiration[17];
   char  volumeEffective[17];
```

```
   char  FileStructureStandardVersion;
   char  Reserved4;           /* Must be 0. */
   char  ApplicationUse[512];
   char  FutureStandardization[653];
} PVD, *PVDPtr;
```

The path table looks like this in C:

```
typedef char      dirIDArray[8];

typedef struct
{
      byte  len_di;          /* Length of directory identifier. */
      byte  XARlength;       /* Extended attribute record length. */
      Long  dirLocation;     /* First logical block where directory is stored. */
      Word  parentDN;        /* Parent directory number. */
      dirIDArray  dirID;     /* Directory identifier: actual length is */
                             /* len_di; there is an extra blank */
                             /* byte if len_di is odd. */
} PathTableRecord, *PathTableRecordPtr;
```

Notice that this strucure is difficult to describe in C, because C requires that arrays of characters have a fixed size, and the character arrays in these records are variable in size. The path table records are packed together, so you'll see some grungy code to move a pointer along in the variable records of the path table.

The directory record looks like this in C:

```
typedef struct
{
      char  signature[2];      /* $41 $41 - 'AA' famous value. */
      byte  extensionLength;   /* $0E for this ID. */
      byte  systemUseID;       /* 02 = HFS. */
      byte  fileType[4];       /* Such as 'TEXT' or 'STAK'. */
      byte  fileCreator[4];    /* Such as 'hscd' or 'WILD'. */
      byte  finderFlags[2];
} AppleExtension;

typedef struct
{
      byte  len_dr;            /* Directory record length. */
      byte  XARlength;         /* Extended attribute record length. */
      Long  lsbStart;          /* First logical block where file starts. */
      Long  msbStart;
      Long  lsbDataLength;     /* Number of bytes in file. */
      Long  msbDataLength;
```

```
        byte  year;             /* Since 1900. */
        byte  month;
        byte  day;
        byte  hour;
        byte  minute;
        byte  second;
        byte  gmtOffset;        /* 15-minute offset from Universal Time. */
        byte  fileFlags;        /* Attributes of a file or directory. */
        byte  interleaveSize;   /* Used for interleaved files. */
        byte  interleaveSkip;   /* Used for interleaved files. */
        Word  lsbVolSetSeqNum;  /* Which volume in volume set contains this file. */
        Word  msbVolSetSeqNum;
        byte  len_fi;           /* Length of file identifier that follows. */
        char  fi[37];           /* File identifier: actual is len_fi. */
                                /* Contains extra blank byte if len_fi odd. */
        AppleExtension apple;   /* This actually fits immediately after the fi[] */
                                /* field, or after its padding byte. */
} DirRcd, *DirRcdPtr;
```

Again, this structure is difficult to describe in C. The directory records are packed into 2048-byte blocks. No directory record is allowed to span a block, so any extra bytes at the end of a directory record block are ignored. We'll ignore such details in this simple example.

Our basic flow of control is simple. The core of the program is in the file BuildISO.c. (See **CreateAVolume** for the main core code.) When we get a floppy, we check to see if it is formatted. If so, we ask the user if he or she wants to continue (to make sure we don't accidentally destroy a useful floppy). We create a primary volume descriptor (by calling **CreatePVD**) and fill in most of the fields with blanks. We create a simple path table. Because we don't have any subdirectories, we can build an extremely simple path table with only one entry (for the root). We make a copy of the path table in both least-significant-byte and most-significant-byte order.

At this point, we loop, prompting the user for a filename. (See the routine **CreateFiles** for details.) When the user selects a file, we get the Finder information for that file (**GetFileInfo**) and check to see if the file has a resource fork. If the file has a resource fork, we create an associated file directory record, and copy the resource fork to the floppy. We always create a regular file, even if the file in question has no data fork. (This is an arguable point. The Macintosh ISO 9660 support works fine on files with only an associated file, but users of other operating systems get bothered by the fact that files consisting of only an associated file don't show up in their directory listings. Creating a regular file, even if the data fork is empty, ensures that the same number of files shows up on the Macintosh and MS-DOS or other operating systems.)

POSSIBLE IMPROVEMENTS

Improvements you can make to this sample program include the following:

- Add secondary volume descriptor support.
- Add subdirectories. There are a lot of subtle issues with ordering of records in the path table that I've managed to avoid by not permitting subdirectories.
- Give the program a real user interface. Ideally, you'd show a Finder-like set of volumes and let the user drag files (and folders) from a HFS volume onto the ISO 9660 volume.
- Allow hard disks to be specified. This requires changing the **`drvName`** constant in iso9660.h.

IN CONCLUSION

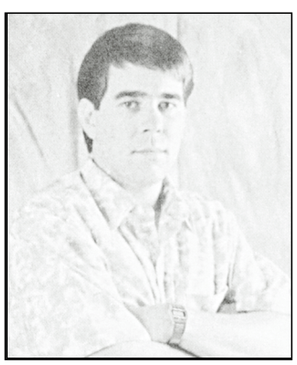
If you've read to this point, you know more about ISO 9660 and High Sierra than you ever thought your attention span could tolerate. You know where the formats came from, how they're implemented on the Macintosh, what they specify, how Macintosh files are stored in these formats, how to press a CD-ROM in one of these formats, and even how to write a program to convert HFS files to one of these formats.

The point of all this is that ISO 9660 (and its older cousin High Sierra) gives you an operating system independent platform for delivering information, thus opening up new markets for your applications. If you are trying to penetrate multiple markets without using ISO 9660, you are just pounding sand.

Thanks to Our Technical Reviewers:
Bill Galcher, Matt Gulick, Andy Poggio,
Llew Roberts, Keith Rollin, Helen Wang

HOW TO CREATE A MIXED-PARTITION CD-ROM

Since the original Phil & Dave's Excellent *CD was released, containing both Macintosh HFS and Apple II ProDOS volumes, DTS has gotten many questions about how it was done. Some ask just out of curiosity, while others want to create their own mixed-partition CD-ROMs. This article gives a detailed account of how any developer can prepare a mixed-partition hard disk whose image can be pressed onto CD-ROM.*



LLEW ROBERTS

The process of producing a CD-ROM disc containing both HFS and ProDOS volumes is relatively simple and straightforward. It's facilitated by the fact that Apple's operating systems recognize the data track of a CD-ROM as if it were a SCSI hard disk. You prepare a hard disk exactly as you wish it to appear on CD-ROM, ship it off to a CD production company, and they send you back a CD.

Mixing partitions is easiest if you have a 600+ MB hard disk, but you can also mix partitions if you have two or more smaller hard disks. We'll get down to the brass tacks of this procedure after a preliminary discussion of why you might want to mix HFS and ProDOS partitions, and some background information about partitions that you need to know if you're to fully understand the procedure.

WHY MIX HFS AND PRODOS PARTITIONS?

Why would a developer want to create a CD-ROM that mixes HFS and ProDOS volumes? For one thing, combining HFS and ProDOS volumes on one CD is a way for developers of Apple II applications to make their applications and files available both locally and through AppleShare® fileservers (which only read CD-ROMs in HFS format) with minimal additional effort. For another, mixing partitions is a way to distribute applications and files so they can be read by both the Macintosh and the Apple II.

LLEW ROBERTS became an Apple person by accident, although we're not sure exactly which accident that was. (Lately there have been several.) He says he works on just too many different things to actually specify what he does for a living. We think he may be a DTS engineer, because he was recently overheard answering questions regarding mixing HFS and ProDOS partitions on a single drive. His only REAL hobby is collecting originals and English translations of Japanese manga and anime (comic books and animated video). His favorite is AppleSeed. Llew also dabbles in subliminal suggestion. Can you find the hidden message in his article? You'll know for sure when you awaken in the middle of the night craving some manga or a CD-ROM drive. •

Perhaps a more obvious solution to the problem of creating a CD-ROM readable by both the Macintosh and the Apple II would be to convert all volumes to the ISO 9660 format, described in Brian Bechtel's article in this issue. A CD-ROM in ISO 9660 format is readable not only by both the Macintosh and the Apple IIGS but by other operating systems as well.

For developers who rely on CD-ROM to store large amounts of information accessible by only one application, this is adequate and in some cases preferred. On the other hand, for developers who wish to use CD-ROM to distribute multiple applications, graphics and sound files, or other files that the user can browse through and launch using the Finder (such as Apple's Developer CD Series and the public domain CDs being released by user groups), using the ISO 9660 format presents certain problems.

Storing files in ISO 9660 format strips the Finder of its ability to read desktop information about those files. On the Macintosh desktop, applications on ISO-format CD-ROMs are shown as generic application icons, documents as generic document icons, and folder and file placement information is lost. In addition, although there are supported extensions to ISO to handle Apple IIGS filetype and auxtype information, software is not available at this time to apply these extensions before pressing.

Another reason for choosing to use native file formats rather than ISO 9660 is that conversion into the latter format involves an additional step in the process of pressing a CD-ROM: a premastering system must be used to create a tape that a production company then uses to create a CD-ROM. If your CD-ROM will be used only on an Apple computer, there's really no need for you to go through this additional step.

ABOUT PARTITIONS

Partitions are logical volumes on a hard disk.

ProDOS is limited to 32 MB volumes, so under the ProDOS file system, a 20 MB hard disk would usually have only one ProDOS volume on it, while a 650 MB hard disk would probably be partitioned into several ProDOS volumes. HFS can handle very large volumes, so there is rarely a need for more than one HFS volume on one disk. (Note that the Macintosh driver currently supplied with Apple HD SC and CD SC drives will support *only one* HFS partition. Most large third-party drives will support multiple HFS partitions. Apple does not recommend shipping CD-ROM with multiple HFS partitions.)

A disk is partitioned and the partitions are initialized with software that is included on the system disks or with the hard disk drive. Advanced Disk Utility (ADU) for the Apple IIGS, included with System Disk 5.0 or later, will satisfy most Apple II

partitioning needs. It supports all drives that follow the Apple extensions to the ANSI SCSI standard, and most that follow the ANSI SCSI standard faithfully, even without the Apple extensions. For the Macintosh, partitioning software is usually included with a hard disk drive.

SCSI hard disks store block allocation information (that is, number and size of the partitions and drivers on the disk) in the first few physical blocks of the disk. The hard disk driver creates logical volumes from this information at boot time and mounts these partitions as volumes on the desktop. Figure 1 illustrates the layout of a typical hard disk with mixed partitions.

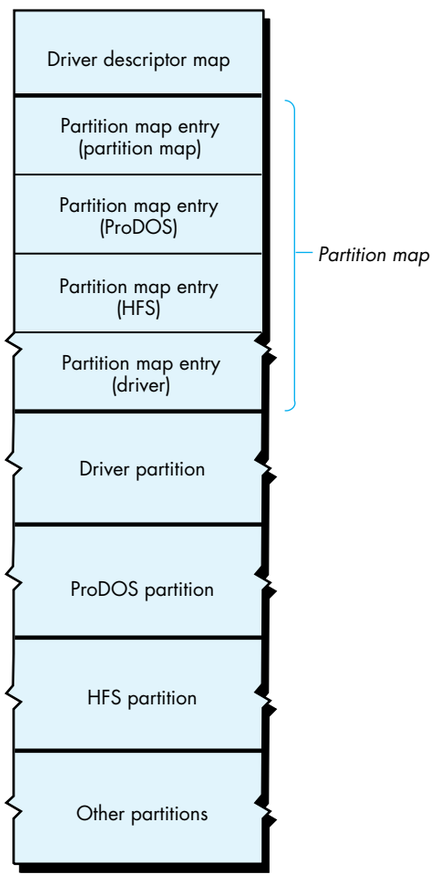


Figure 1
The Layout of a Typical Hard Disk With Mixed Partitions

Physical block 0 of the disk contains the driver descriptor map (DDM), which describes the drivers on the disk. When the disk is mounted, this information is used to load the necessary drivers, as detailed in *Inside Macintosh*, volume V, page 576. The Macintosh requires that a driver be resident on the disk; the Apple II supports drivers if they are resident on the disk, while not requiring them to be.

Starting at physical block 1 of the disk is the partition map. Each partition on the disk is described in its own partition map entry (PME) in this partition map. A PME, which occupies one block and is built when the partition is initialized, consists of a series of data fields describing the size and state of a specific partition. With the exception of physical block 0, every block on the disk must be accounted for in a PME, as belonging to a partition. The partition map is itself a partition and contains a PME describing itself. The PME format is shown in Figure 2.

$00	pmSig (word)	*Always $504D*
$02	pmSigPad (word)	*Reserved for future use*
$04	pmMapBlkCnt (longword)	*Number of blocks in map*
$08	pmPyPartStart (longword)	*First physical block of partition*
$0C	pmPartBlkCnt (longword)	*Number of blocks in partition*
$10	pmPartName (32 bytes)	*Partition name*
$30	pmPartType (32 bytes)	*Partition type*
$50	pmLgDataStart (longword)	*First logical block of data area*
$54	pmDataCnt (longword)	*Number of blocks in data area*
$58	pmPartStatus (longword)	*Partition status information*
$5C	pmLgBootStart (longword)	*First logical block of boot code*
$60	pmBootSize (longword)	*Size in bytes of boot code*
$64	pmBootLoad (longword)	*Boot code load address*
$68	pmBootLoad2 (longword)	*Additional boot load information*
$6C	pmBootEntry (longword)	*Boot code entry point*
$70	pmBootEntry2 (longword)	*Additional boot entry information*
$74	pmBootCksum (longword)	*Boot code checksum*
$78	pmProcessor (16 bytes)	*Processor type*
$88	(128 bytes)	*Boot-specific arguments*

Figure 2
The Format of a Partition Map Entry

In condensed form, the partition map for a hard disk with both HFS and ProDOS partitions looks like this:

Block	pmMapBlkCnt	pmPyPartStart	pmPartBlkCnt	pmPartName	pmPartType
1	6	1	3F	Apple	APPLE_PARTITION_MAP
2	6	40	20	Macintosh	APPLE_DRIVER
3	6	60	10000	/PRODOS.1	APPLE_PRODOS
4	6	10060	10000	/PRODOS.2	APPLE_PRODOS
5	6	20060	28000	MacOS	APPLE_HFS
6	6	48060	D6800	Extra	APPLE_FREE

pmMapBlkCnt is a count of valid PMEs on the hard disk. This longword is contained in each valid PME. If it is modified in one, it must be modified in all PMEs. If a partition has been added manually (that is, with a SCSI block editor) and is not recognized by the operating system, the cause is usually an incorrect value in **pmMapBlkCnt**.

pmPyPartStart is the address of the first physical block of the partition. If the first physical block of a partition (logical block 0) is at physical block $200 of the hard disk, then reading block $20 of the partition actually reads physical block $220 of the disk.

pmPartBlkCnt is the size, in blocks, of the partition. The size of the last partition on the disk is arrived at by subtracting the address in **pmPyPartStart** for this partition from the total number of blocks on the disk.

pmPartName is the name of the partition. It serves to identify the partition and should not be confused with the volume name.

pmPartType is the partition type and can contain (but is not limited to) the following:

APPLE_DRIVER	Partition contains a device driver
APPLE_PARTITION_MAP	Partition contains a partition map
APPLE_SCRATCH	Partition is unused and free for use
APPLE_HFS	Partition contains Macintosh HFS volume
APPLE_PRODOS	Partition contains Apple II ProDOS volume
APPLE_FREE	Partition is unused and unusable

APPLE_SCRATCH partitions are areas of the disk that are currently unused, but that can be recognized and initialized by the operating system. In the process of creating a mixed-partition disk, this is the type to assign to partitions that will later be initialized in ProDOS format (assuming that HFS partitions are formatted first). **APPLE_FREE** is the type to assign to partitions consisting of blocks that will not be

used but must be accounted for in order to fulfill the requirement that all blocks on the disk belong to a partition.

When you go about mixing partitions, as described in the following section, you may need to change some of the fields in a PME, and to copy blocks from one disk to another. PMEs can be browsed and edited with SEDIT, a utility written at Apple by David Shayer. Figure 3 shows a PME viewed in SEDIT. This utility also makes it easy to perform block editing at a device level on SCSI hard disks. SEDIT can copy blocks on the same or between separate devices, and provides nifty templates for editing blocks of data. You'll find SEDIT included, along with documentation, on the *Developer Essentials* disc. (A word to the wise: SEDIT also has the wonderful ability to scramble any SCSI device that is connected, so be sure to look at the warning message under the File menu and to read the documentation before trying anything you're not sure of.)

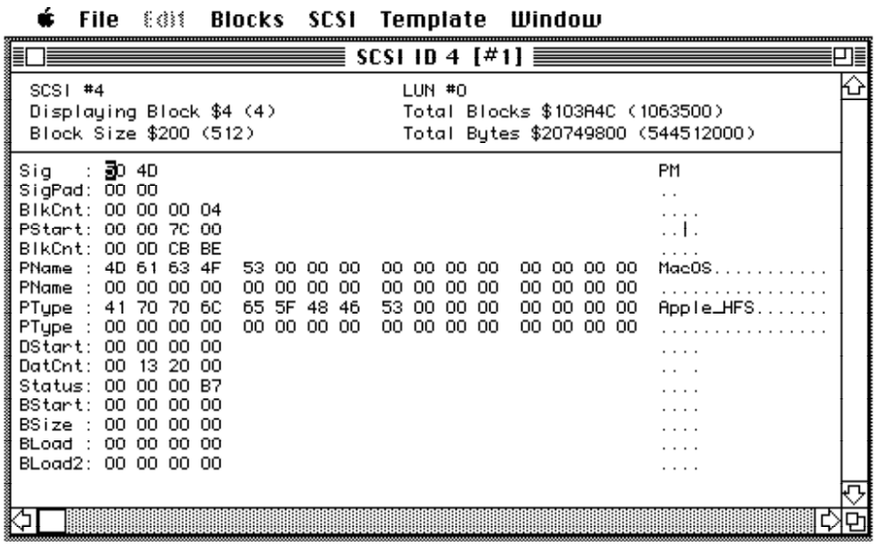


Figure 3
SEDIT View of the PME for the HFS Partition of *A Disc Called Wanda*

THE PROCEDURE FOR MIXING PARTITIONS

Now that you understand the layout of the disk and the importance of the partition map, you're ready to mix your own partitions. You can choose to either include the same information on both partitions (for example, large databases) or arrange the files so that Apple II-specific information is on a ProDOS volume and Macintosh-specific information is on an HFS volume. And you have a choice of whether to start out with one large hard disk, or two or more smaller hard disks. The first way is easiest.

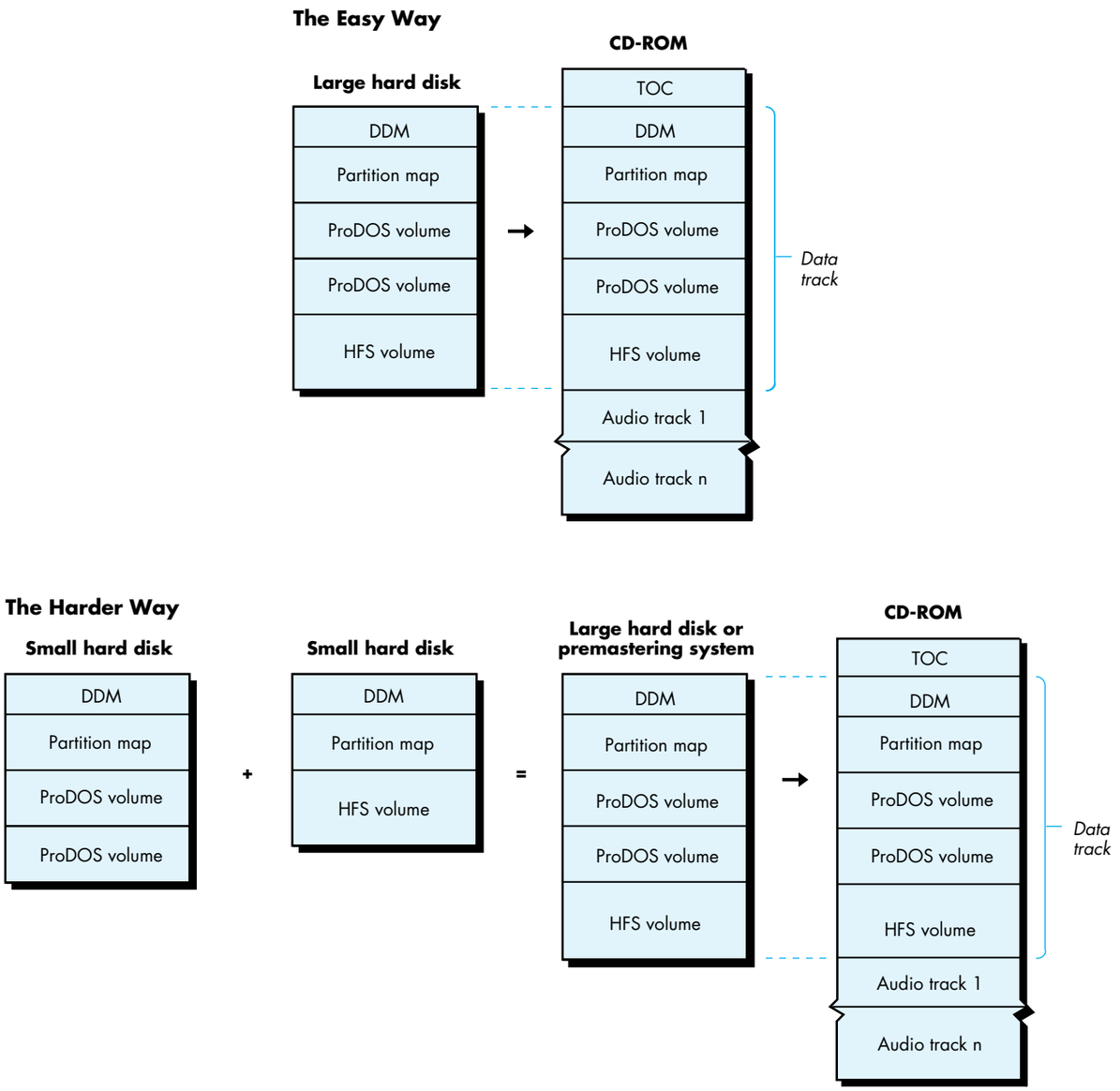


Figure 4
Two Ways to Create a CD-ROM

In the processes described here, every attempt has been made to let the existing system software and utilities do the work, with a minimum of "twiddling" necessary by the developer. This ensures that the CD-ROM will work properly and will be compatible with future system software.

MIXING PARTITIONS ON A LARGE HARD DISK

The simplest method to prepare a mixed-partition disk from which to press a CD is as follows:

1. Beg, borrow, steal, or even (gasp!) buy a 600+ MB hard disk drive that will work with both the Apple IIGS and the Macintosh. Software to partition the hard disk for the Macintosh is usually included with the drive.

2. Use the Macintosh partitioning software on the hard disk. This will create the DDM in block 0. Create an **`APPLE_SCRATCH`** partition for each ProDOS volume you wish to include on the disk. Remember that ProDOS volumes are limited to 32 MB and that only the first two volumes will be accessible under ProDOS 8 (but all will be accessible under GS/OS). Make the HFS partition the last one on the disk, to allow for changing the size of this partition without disturbing the ProDOS partitions.

   If you create two **`APPLE_SCRATCH`** partitions, the partition map will look something like this:

Block	pmMapBlkCnt	pmPyPartStart	pmPartBlkCnt	pmPartName	pmPartType
1	6	1	3F	Apple	APPLE_PARTITION_MAP
2	6	40	20	Macintosh	APPLE_DRIVER
3	6	60	10000	Scratch	APPLE_SCRATCH
4	6	10060	10000	Scratch	APPLE_SCRATCH
5	6	20060	28000	MacOS	APPLE_HFS
6	6	48060	D6800	Extra	APPLE_FREE

3. Disconnect the drive from the Macintosh (with the power off, of course), and connect it to the Apple IIGS. Boot the system with System Disk 5.0.2 or later (to take advantage of the new SCSI Manager and drivers). When the Finder's desktop appears, a dialog will be presented declaring that the disk is unreadable. Click Initialize for each of the **`APPLE_SCRATCH`** partitions (that's twice for the above example).

Warning: Do not initialize the HFS partition! The Finder will also want to initalize the HFS partition, since it doesn't recognize it, and you may politely decline by clicking Eject in the dialog box.

Figure 5
The Dialog Box to Initialize Partitions

Every time the GS Finder is launched (booting, quitting from an application, and so forth) it will ask if you wish to initialize the HFS partition. This annoying behavior will disappear when the CD-ROM with the image of the hard disk is mounted, since it is write-protected.

The hard disk is now fully prepared. In our example, it contains two ProDOS volumes and one HFS volume, which are fully initialized and ready for files to be copied onto them.

4. Copy the desired files to their respective volumes. Transfer the hard disk drive between the Apple IIGS and the Macintosh as needed, until the files and folders for all volumes are arranged as you wish them to appear on the CD.

5. Mail your hard disk drive to the CD production company of your choice, asking them to place an image of the hard disk on the data track of the CD.

MIXING PARTITIONS FROM SMALLER HARD DISKS

In cases where you wish to combine partitions from separate hard disks on one large hard disk, more work is required, but it is certainly not impossible.

The same process I'm about to describe can also be done using a CD-ROM premastering system that allows block manipulation by a Macintosh, but you will definitely need technical assistance from the premastering system's engineer. With such a system, it is possible to manually create partition maps and block copy the desired volumes over. The end result is an image of a large hard disk identical to the image achieved by the process described in the preceding section and below.

As an example to illustrate the process of combining smaller hard disks on a larger one, let's say we're starting with two hard disks—one 80 MB hard disk formatted as a large HFS volume and one 80 MB hard disk with two 32 MB ProDOS partitions. The partitioning utilities and system software have already done most of the work for us: the partitions are initialized and the partition maps built.

The partition map for our first hard disk, SCSI ID 1, looks like this:

Block	pmMapBlkCnt	pmPyPartStart	pmPartBlkCnt	pmPartName	pmPartType
1	4	1	3F	Apple	APPLE_PARTITION_MAP
2	4	40	20	Macintosh	APPLE_DRIVER
3	4	60	2626E	MacOS	APPLE_HFS
4	4	262CE	4	Extra	APPLE_FREE

Note that because not all hard disks, even of the same capacity, have the same block count, the value in **PmPartBlkCnt** for the last partition could differ if a different hard disk were being used.

The partition map for our second hard disk, SCSI ID 2, looks like this:

Block	pmMapBlkCnt	pmPyPartStart	pmPartBlkCnt	pmPartName	pmPartType
1	4	1	3F	Apple	APPLE_PARTITION_MAP
2	4	40	10000	/PRODOS.1	APPLE_PRODOS
3	4	10040	10000	/PRODOS.2	APPLE_PRODOS
4	4	20040	6292	Extra	APPLE_FREE

Manually combining the partition maps on paper, we come up with the desired partition map for the large hard disk, SCSI ID 3:

Block	pmMapBlkCnt	pmPyPartStart	pmPartBlkCnt	pmPartName	pmPartType
1	6	1	3F	Apple	APPLE_PARTITION_MAP
2	6	40	20	Macintosh	APPLE_DRIVER
3	6	60	10000	/PRODOS.1	APPLE_PRODOS
4	6	10060	10000	/PRODOS.2	APPLE_PRODOS
5	6	20060	2626E	MacOS	APPLE_HFS
6	6	462CE	D8592	Extra	APPLE_FREE

What remains is to combine all of the partitions on the third hard disk. To do so, we first copy block 0 (the DDM) from hard disk 1 to hard disk 3. Then we copy the partition map from hard disk 1 to hard disk 3. We copy physical blocks 3 and 4 from hard disk 1 to physical blocks 5 and 6 on hard disk 3.

From hard disk 2, we copy physical blocks 2 and 3 to blocks 3 and 4 on hard disk 3. The final partition map is now in place, although the values in some of the fields are incorrect. We use SEDIT to update the fields according to the manually created table, remembering to update **`pmPyPartStart`** in each entry and **`PmPartBlkCnt`** for the last partition on the disk (to adjust for the changed number of unused blocks in the **`APPLE_FREE`** partition). Now hard disk 3 is ready to have the volumes copied to it.

Using the SEDIT Copy Blocks command, we copy the volumes from the smaller hard disks to the proper locations on the large hard disk:

	From SCSI ID	To SCSI ID	From Block	To Block	# of Blocks	
1.	1	3	0	0	1	DDM
2.	1	3	40	40	20	Mac Driver
3.	1	3	60	20060	2626E	HFS Partition
4.	2	3	40	60	10000	ProDOS partition
5.	2	3	10040	10060	10000	ProDOS partition

After we copy the DDM from hard disk 1 to hard disk 3 it is no longer valid, so we zero out the DDM's first 24 bytes. (If we planned to use hard disk 3 from the Macintosh and not as a master for a CD-ROM, we would update these bytes to make the DDM valid for hard disk 3.) We also zero out the first 8 bytes (the boot block) of the HFS partition to ensure that the CD doesn't attempt to boot.

The large hard disk is now fully prepared and ready for shipment to the CD production company.

ODDS AND ENDS

If after reading this article you're eager to try creating your own mixed-partition CD-ROM, you'll want to refer to the sidebar on CD-ROM production companies in Brian Bechtel's article in this issue. There you'll find names and addresses of places to send your hard disk. Brian's paper *CD-ROM and the Macintosh Computer*, found on the accompanying *Developer Essentials* disc, covers basic details of cost and time required to get a CD-ROM pressed.

And here's a final note to round out your understanding of mixed-partition CD-ROMs. When the CD-ROM resulting from the process described in this article is mounted, the partitioned volumes on its data track are recognized and mounted on the desktop. The Macintosh will currently mount only the first HFS partitioned volume that it finds. The Apple II will try to mount the HFS volume but will not find a file system translator to read it with, and so will effectively ignore it.

Thanks to Our Technical Reviewers:
Bryan Atsatt, Matt Gulick, Jim Luther, Dave Lyons, Jim Reekes, Dave Shayer

MACINTOSH Q & A

Q

How can I keep track of a file the next time my application is launched?

A

Technical Note #238, Getting a Full Pathname, documents the recommended method for "remembering" a file's location.

> . . . you should remember the DirID of the directory the file is in along with its name. This way, you will still be able to find your file even if the directory has been moved. Under System 7.0 or later, save the file's unique 32-bit ID number as well, so that you can also find the file even if its name has changed.

To remember a file's location, keep the volume name, DirID, and filename. This information is all you need to locate any file. Standard File returns the DirID of the file in CurDirStore or the wdRefNum in the vRefNum field of the reply.record. Note that Technical Note #238 mentions how to get a file's DirID while in Standard File. Given the working directory, you can find its vRefNum and DirID by calling _GetWDInfo. Refer to *Inside Macintosh*, volume IV. Volume references and working directories are dynamic; they change every time the system is booted, so you cannot use the vRefNum or wdRefNum. Typically, the volume name and filename are not changed. The DirID will not change unless the user deletes the folder. Renaming the folder does not change its DirID.

First ask the user to locate the file by calling SFGetFile. Keep the volume name, DirID, and filename for this file. The next time you want to locate the file, use this same information. If you do not find the file, then again call SFGetFile asking the user to locate it.

DTS has an example application, SC.018.StdFile, which you may find helpful. You can find this in the Sample Code folder on the enclosed *Developer Essentials* disc.

Q

How can I determine the size of my application's MultiFinder partition?

A

It's really difficult to find the exact size of the memory partition that the application is running under. If it can be determined, I doubt that the effort would be worth the trouble. I think the real concern you have is the size of the available stack and heap, but not the entire partition. Since there is little that an application can do to change its partition size (except to change the 'SIZE' resource and then force a relaunch), the real concern would be to find the size of the available stack and heap. Included in the application's partition are the application parameters, jump table, application globals, and QuickDraw globals. The size of the partition is not easily determined. The only portions of an application's memory use that are adjustable at run time are the stack and the heap.

These questions and answers are compiled by the Macintosh Developer Technical Support group.

The stack and heap sizes are fixed within the boundaries of the entire application partition. Increasing one decreases the other. There are Memory Manager calls to change the size of the heap. To increase the stack size, you decrease the heap's size.

Q

In earlier versions of the Chooser, there was a limit of 16 volumes per server for AppleShare servers. Has this limit changed in System 6.0.4?

A

The limit of 16 volumes per server in the Chooser has not changed with System 6.0.4. We hope to have a new version of the Chooser for System 7.0.

Q

How do I force the Finder to update its windows after my application has changed a file's FndrInfo?

A

There is no direct way to tell the Finder to update the desktop. The Finder will synchronize the desktop file's appearance after it detects that the volume's modification date has changed. Whenever you create or delete a file, or move it to another folder, the hierarchical file system (HFS) will change the modification date of the volume and that folder. When the Finder has noticed the volume's modification date has changed, it begins scanning about once every 10 seconds for changes in all of the open folders.

Changing the file's FndrInfo or renaming it is not going to change the modification date. As a suggestion for an installer program, you can initially create a temporary file. Once all the files are installed you can delete the temporary file. Deleting this temporary file as a last step will cause the Finder's window to be updated.

Q

My little application has two handles in memory that have been allocated. I want to lock one handle high in memory and the other one low in memory. I noticed that the Mac toolbox has the functionality to lock a handle high (MoveHHi); however, I did not notice any routine that would move the block low in memory, before a lock. I'm looking for a MoveHLow routine. Does one exist? If not, how would I go about doing this?

A

There is no similar functionality for locking a handle low. The best way to go about doing this is to use NewPtr, which automatically allocates the block as low as possible. Of course, it's not a handle, but it's still a locked block as low in the heap as possible.

Another way to do this is to use ResrvMem which, as Inside Macintosh, volume 2, page 39 says, "will try every available means to place the block as close as possible to the bottom of the zone, including moving other blocks upward, expanding the zone, or purging blocks from it." Then make your call to NewHandle with the same size as requested in ResrvMem. That'll allocate the handle as low as possible.

Q

How can I support multiple HFS partitions on a SCSI device?

A

If at all possible, avoid trying to support partitions. We'll warn you up front that an ejectable drive that contains multiple HFS partitions is not going to be anything less than difficult. You'll be better off not attempting to support multiple HFS partitions. It greatly complicates the code, and there are user interface problems too. What if the user ejects one of the partitions? What should happen? This is technically difficult for the driver to handle.

If the user ejects a partition, then the driver might eject the media and mark all of its remaining partitions as off-line. If the user drags a partition to the trash, this should unmount only that partition (but then how would the user unmount the entire media?). The remaining partitions should be marked off-line and the user will see them as gray icons on the desktop. If users want to access one of these partitions, they'll get the Disk Switch alert. They need to insert the proper cartridge and the device will then post a disk insert event for every partition (because it cannot determine exactly which partition is really needed). This will again bring all partitions back. The trap _Offline should take care of all this for you, but it cannot be called at interrupt time. Therefore, the driver will need to use accRun calls to use _OffLine.

Again, the system doesn't support multiple HFS partitioned drives. It only expects to find one HFS partition on a volume. The system will attempt to read from the first HFS partition and then stop. If the first one is not bootable, then that device cannot be a startup device. If you attempt to put more than one partition on a device, then you have to perform additional hacks to mount them. Be warned that hacking this feature into your drive involves a compatibility risk.

All the work will be up to the driver. It will have to find the extra partitions and mount them. Each partition will have a drive queue entry having each element reference the same driver. When your driver's open routine is called, you call _AddDrive for each partition. This calls _Enqueue and installs each element into the drive queue. Once the driver is closed, you should remove each of the queue elements with _Dequeue.

Q

I would like to write James Brown in jail, but now that he is on work release, where do I write?

A

You can write the Godfather of Soul at

Lower Savannah Work Center
Route 4, Box 50
Aiken, SC 29801

Brown is serving concurrent six-year and six-year-and-three month terms for his involvement in a wild, two-state car chase in September of 1988. He won't be eligible for parole until 1992.

APPLE II Q & A

Q

In the previous issue of develop, an answer said that anyone doing a Close with reference number zero will close New Desk Accessory resource forks. Does this mean my NDA can't use resources?

A

No. It means that your NDA can't open its resource fork at DAInit time and expect it to always be open. You can use resources in an NDA by opening the resource fork when your DA is opened and closing it when your DA is closed. Although an application could still close your resource fork while your DA window is open but not active, this is not likely.

Q

Matt, I found your caching article to be very informative, but I'm confused about the driver level of caching. You say that if the cachePriority word on GS/OS direct page is zero, the driver should not write the block to the cache. If the block was already in the cache, won't this mean the disk has a different block than is in the cache, messing things up drastically?

A

Thanks, and yes. When a driver is supposed to ignore the cache on a write call (because the cachePriority field is zero), it still must deal with the possibility that the block may already be in the cache. When writing with cachePriority zero, drivers may not add new blocks to the cache but must update blocks that are *already* in the cache. This step is necessary because the next time someone reads with caching enabled, the block in the cache will be returned. Next time I forget something, I promise that it will be less important.

Q

I did a SelectMember2 on an extended list control, and the list was drawn at a funny place in my window. What gives?

A

The current port was not set to the window that the list control was in. Most List Manager calls, and many other toolbox calls, require that the current grafPort be explicitly set. Before you call SelectMember2, set the current port to your window with a SetPort call. Remember the note in the *Apple IIGS Toolbox Reference*, volume 2, under the NewWindow call—"Important: NewWindow does not set the current port, but many routines require that a current port exist. Use the QuickDraw II routine SetPort to set the current port." Using SetPort can prevent toolbox confusion and reduce your debugging time.

Q

Someone told me that there's an easier way of keeping track of double and triple clicks in System Software 5.0 and later. What is it?

A

The extended Task Record introduced in System Software 5.0 includes a new field called wmClickCount, at offset $1C. If you're using TaskMaster in your application, set bit 19 of your wmtaskMask (the tmMultiClick bit)

These questions and answers are compiled by the Apple II Developer Technical Support group. •

and TaskMaster will keep the wmClickCount field updated. Every time there is a mouse-down event within the limits set by the user in the control panel setting 'Double Click', TaskMaster will increment wmClickCount by 1. If you're interested in double or triple clicks, check wmClickCount on every mouse-down-related event. If it's at 2, then a double click was the last mouse-down result, three, it was a triple click. In fact, you can track as many closely spaced clicks as you like—quintuple, sextuple, on up—if you really want to get silly. Apple recommends not going further than triple-clicking since more clicks become quite unwieldy.

Q

What the heck is the resource ID for a control color table, and why is it so hard to find in the Types.Rez interface file?

A

The resource ID for control color tables is $800D, and is referenced in Types.Rez as resource name r_BBBB. It is listed under its regular name (rCtlColorTable) in Appendix E of the *Apple IIGS Toolbox Reference*, volume 3. The obscure name in Types.Rez and the lack of a resource structure either in Types.Rez or Appendix E is caused by the structure being variable, depending on the control type associated with the color table.

Q

ACE compression and expansion don't work consistently for me. I can compress (or expand) one sound correctly, but the next time I try it the results are all wrong. What gives?

A

You forgot to call ACECompBegin or ACEExpBegin between different compressions or expansions. ACE maintains information on the current operation in its direct page space to allow you to do multipart expansions or compressions. When you start working with a new sound sample, you have to call either ACECompBegin or ACEExpBegin to inform ACE that you're starting a new operation, and all the internal data should be reset. More information can be found in the ACE chapter of the *Apple IIGS Toolbox Reference*, volume 3.

Q

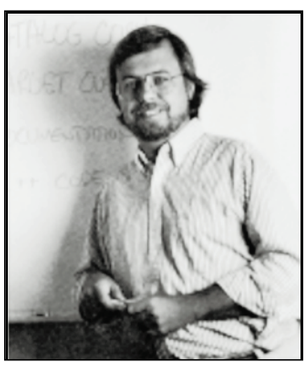
Do I need to purchase APW Tools & Interfaces if I already bought Programming Tools & Interfaces for APW?

A

Yes! The original *Programming Tools & Interfaces* utilities did not correctly support ORCA/Pascal and ORCA/C. Also, every tool in the package has been updated and enhanced. A new C library, used to build some of the tools, had bugs fixed in it (and will be available separately later). The new package also includes completely revised Apple IIGS interfaces for APW C and the APW/ORCA assembler. The new *APW Tools & Interfaces* package is a class 1 product that may be ordered from Developer Tools Express; the previous class 2 product is no longer supported.

DEVELOPER ESSENTIALS: ISSUE 3

Scott Converse, Corey Vian, Cleo Huggins, and Mary Skinner put develop in electronic form. Read more about the Electronic Media Group below.



The allegedly 27-year-old Jack Hodgson, product manager of *Developer Essentials*, produced and directed corporate videos in Boston, ran a small, computer book publishing company, did some free-lance programming, and founded the Boston Computer Society's Mac Users Group. His next big life goals are to buy his own plane and to learn to play his piano well enough to cut loose in Dave Szetela's Excellent Annual WWDC Moofamania Jam Sessions (caution: unofficial title).

Introducing the new Developer Essentials *disc. In addition to* develop *and related code, on this issue of the disc you'll find tools and information we think every developer should have. These pages highlight what's on the disc, but once you start browsing, you'll also find a few surprises.*

To use the disc, you need a CD-ROM drive and the appropriate cables and connectors. Refer to your CD-ROM drive's owner's manual for detailed information about connecting the drive to your particular machine.

For a Macintosh, you need at least 1 MB of memory, System 4.1 or later, and Finder 5.4 or later. In addition, you need to copy the Apple CD-ROM INIT that comes with the CD drive startup disks into your System Folder. For an Apple II, your SCSI card must have Rev C or later ROM. With ProDOS, no special setup is required. If you use GS/OS, you must use the Installer on System Disk 4.0 or later to install the CD-ROM driver onto your startup volume.

SCOTT CONVERSE is the group's Electronic Media Mogul and leader. A true on-line addict, he makes a living cruising the electronic highways and getting information to as many people as possible by using computers. Scott also loves sci-fi (particularly cyberpunk), reads books on design, and plays music on any of six full-blown, wall-shaking stereo systems in his house. When not cruising the electronic highways, he's racing radio-controlled cars. Would you ride the fiber optic byways with this guy?

COREY VIAN takes the Zen approach to most things. He has an interdisciplinary B.A. in art and math from Maharishi International University. (Really! It's in Iowa.) An eight-month Apple veteran (two years and eight months if you count his prior consulting), he's now doing information interface design. (See his sidebar in the first article in this issue.) An avid meditation practitioner, he also flies airplanes, builds cabinetry, windsurfs, snow skis, practices aikido, and composes R&R music—and he claims he isn't busy.

develop
You've read the articles, you've bought the arguments, and now it's time to write your own code. The idea is that you don't waste your time typing the example programs—just mount this handy CD-ROM, then copy and paste. We've included develop as well as the code from each of the articles to help you avoid typos. So, browse around, take what you need, and save the rest for a rainy day. Each new issue of *Developer Essentials* will archive all of the back issues of the journal and the code. So look forward to one-stop searching coming soon to a CD-ROM near you. If you don't yet have a CD-ROM drive, you should be able to find the contents of *Developer Essentials* on AppleLink, the Apple FTP site on the Internet, and other on-line services in the near future.

International System Software
Developer Essentials includes all the latest international versions of Macintosh system software as well as the latest U.S. versions of GS/OS and ProDOS, all in DiskCopy image format. (You must have a Macintosh to run DiskCopy and create floppy disks from these images.)

International HyperCard
Need the latest version of HyperCard? Look no further. *Developer Essentials* includes the latest international versions of this "software erector set" in DiskCopy image format.

DTS Technical Notes and Sample Code
All Apple II and Macintosh Technical Notes and Sample Code programs are included for your reference. Be sure to check here for the latest and greatest development information and Developer Technical Support programming tips and techniques.

Macintosh Technical Notes Stack
This HyperCard stack incorporates all of the latest Macintosh Technical Notes into a single on-line source, which is cross-referenced with *SpInside Macintosh*, Q & A Stack, and the Human Interface Notes Stack.

Macintosh Q & A Stack
Got a tough development question? Try the Q & A Stack, which is a collection of the most frequently asked questions DTS receives from developers. Organized by subject, this stack answers the questions within and includes cross-references to *SpInside Macintosh* and the Macintosh Technical Notes Stack.

SpInside Macintosh
Of course the most essential of all documentation for Macintosh developers is *Inside Macintosh*, so *Developer Essentials* offers you *SpInside Macintosh*, an on-line version of volumes I-V. *SpInside Macintosh* combines all five volumes into a single, searchable electronic form that is cross-referenced with the Macintosh Technical Notes Stack, Q & A Stack, and Human Interface Notes Stack.

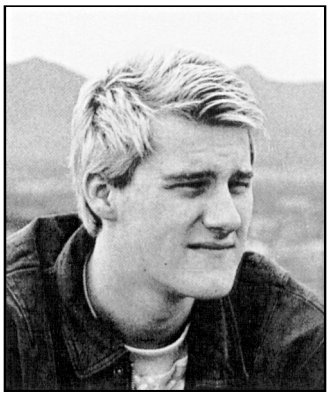
Now you have some of the headliners in *Developer Essentials*, but you should take some time to browse the disc and see what else you might discover. We'll be adding more as *Developer Essentials* evolves, and we hope you agree that these are tools that no developer should be without.

CLEO HUGGINS studied graphic design at the Rhode Island School of Design, taught design and semiotics at the Portland School of Art in Maine, and created the music typeface "Sonata" when she worked at Adobe. She received an M.S. in digital typography from Stanford University, and plays electric violin. Cleo always knew the computer would be a good place to combine her interests; she has joined Apple to help refine the use of typography and design (and maybe even music) in our CDs.

MARY SKINNER collects the input, supervises testing, processes the feedback and is the group's systems administrator (thank goodness Mary is a HyperCard fanatic). She's a native Iowan born in New York City. Her B.A. in physics and B.A. in Russian from the University of Iowa landed her as an Air Force Lieutenant at Johnson Space Center. In her spare time, she plays with the computer, reads sci-fi (she's a Poul Anderson fan), and listens to the nonsoft side of rock and roll. Mozhete skazat "Def Leppard"?

ACCESSING CD-ROM AUDIO TRACKS FROM YOUR APPLICATION

CD-ROM opens up the possibility of providing the user of your application with a new dimension of sound feedback, in full digitally reproduced stereo. Coaxing the sound out of the AppleCD SC® drive, however, is not as simple as prompting the user to "press PLAY on your CD-ROM drive now." This article explains the intricacies of controlling the audio functions of the AppleCD SC from an Apple II application.



ERIC MUELLER

Imagine how you might use CD-ROM audio tracks to make your software burst with sound: language software that pronounces each lesson as it teaches it; reading programs that speak words instead of just displaying them; almost any kind of program adapted with audio cues for an audience with reading disabilities. CD-ROM is also the answer for applications that require lengthy music tracks or background music that simply won't fit on the program disk in a digitized format.

In this article, you'll learn about the capabilities of the AppleCD SC drive, and the kinds of calls you can make to the drive to control the audio features. The article also covers basic information about how audio tracks are stored on CD-ROM. (While the primary focus of this article is the Apple II, this section applies to the Macintosh as well.) Finally, it covers the specifics of playing audio tracks via the GS/OS® SCSI CD driver and the five major audio control calls.

AN OVERVIEW OF THE AUDIO CAPABILITY

You make the AppleCD SC do your bidding by sending it the GS/OS device calls `DStatus` and `DControl` via the GS/OS SCSI CD driver. These calls enable you to control all features of the drive.

You get information about the contents of the disc in the drive and the current status of the drive with two DStatus subcalls: ReadTOC and AudioStatus.

ERIC MUELLER is a free-lance Apple II programmer (with an interest in telecommunications) who leads an unstructured life with no days off, no days on. Despite that, he doesn't seem to have much free time. He plans to go to college somewhere, someday, to study the great unknown. For now, he writes his code, co-manages an Apple II area on GEnie, listens to the B-52's, eats lunch in Chinese restaurants, and watches "Late Night With David Letterman."

During his days off (or is it his days on?) he enjoys teaching "stupid pet tricks" to his two cats, Conan and Aster, and enjoys life in Colorado Springs. Otherwise, he gets perverse thrills by writing Apple II programs that don't go at all " by the book." (He does promise that he absotively, posilutely has memorized the interface guidelines, a MUST in the curriculum of the great unknown.) •

You control audio play with five **DControl** subcalls: **AudioPlay**, **AudioPause**, **AudioScan**, **AudioStop**, and **AudioSearch**. These calls start and stop the disc from spinning inside the AppleCD SC, and position the laser.

We'll look at each of these functions in greater detail in the sections that follow, and illustrate them with code for a CD Remote classic desk accessory (CDA). You'll find the complete source code on the *Developer Essentials* disc in Merlin 16+ format. This code serves three purposes: first, it enables you to experiment with the AppleCD SC drive and see how it responds to certain calls. Second, it documents the exact steps necessary to make audio calls. Finally, you can modify and extend it with your own test code.

TO COMMUNICATE WITH THE DRIVE

Your Apple II application can communicate with the AppleCD SC through calls built into either the SmartPort or the GS/OS SCSI CD driver. Accessing the AppleCD SC's audio features via the Smartport or from the Macintosh side of things is very well documented, while documentation about using the GS/OS SCSI CD driver is not as complete (yet). We'll focus here on how to access the AppleCD SC via GS/OS.

Issuing CD SC commands from your program is a two-step process: first you must locate the drive with a **DInfo** call, and then you can use **DStatus** and **DControl** to check the status and control the device. The control data you send will be parameter lists for the audio calls; the status data you receive will be information about the disc in the drive.

DInfo

Locating the drive with **DInfo** is fairly straightforward: you step through each of the available devices until you find one that has a **deviceIDNum** of $0007 (SCSI CD ROM drive). If your **DInfo** call returns an error $11, that means that you've hit the end of the device chain, and that no CD SC drive is hooked up.

Here is an example from the sample program of how to locate the attached AppleCD SC drive:

```
FindCDRom
         lda   CDROMDev    ;Have we found it before?
         bne   :leave      ;Yes - leave now.

:look                      ;Start looking for drive.
         jsl   GSOS
         dw    DInfo       ;Make GS/OS DInfo call.
         adrl  :devParm
         bcs   :err        ;Leave if error.
```

Complete details on using the SmartPort are given in the *AppleCD SC Developers Guide* (revised edition, Apple Computer, 1989, APDA #A7G0023/A), starting on page 139. Included are parameter lists, a list of all possible CD SC calls, and details on how your parameter list for each of the calls should be set up.

Details on controlling the CD SC from GS/OS can be found in the *GS/OS Reference* (beta draft, Apple Computer, 1988), volume 1 (APDA #A2F2037) and volume 2 (APDA #A0008LL/A). Volume 1, pages 108–109 and 112–119, explains the **DControl** and **DStatus** calls, while volume 2, chapter 2, provides detailed instructions on making each call and gives [continue on next page]

```
         lda   devID      ;Get device ID.
         cmp   #$0007     ;Is it a SCSI CD-ROM device?
         beq   :found     ;Yes - found it.

         inc   devNum     ;No - move to next device . . .
         bra   :look      ;and keep looking.

:found   lda   devNum
         sta   CDROMDev
         sta   DCdevNum   ;Store device number for all control calls.
         sta   DSdevNum   ;Store device number for all status calls.
:leave   clc              ;Found it.
         rts

:none    ~WriteCString #:noCDRom
:1       jsr   getKey
         sec              ;None found!
         rts

:err     cmp   #$11       ;Error $11 - invalid device number?
         beq   :none      ;Yes - no CD-ROM drive found!
         ~WriteCString #:error ;No - some other weird error.
         bra   :1

:error   asc   0d'GS/OS error on DInfo call. Press any key to quit. '0700
:noCDRom asc   0d'No CD-ROM drive found. Press any key to quit. '0700

:devParm dw    8          ;Eight parameters.
devNum   dw    1          ;Device number - start with 1.
         adrl  nameBuffer ;Pointer to buffer for device name.
         dw    0          ;Characteristics.
         dl    0          ;TotalBlocks.
         dw    0          ;SlotNum.
         dw    0          ;UnitNum.
         dw    0          ;Version.
devID    dw    0          ;Device ID: $0007 = SCSI CD-ROM.

nameBuffer dw  31         ;Max length.
         ds    33         ;Storage for device name.
```


detailed parameter lists for each of the CD SC status and control calls.

(Note that on page 64 of the *GS/OS Reference*, volume 2, in the parameter list for the **AudioPause** call, the value for start pause in the pause flag byte [$02] should be $10, not $40.) The *AppleCD SC Developers Guide* gives an explanation of every call. The *GS/OS SCSI Driver (General) External ERS* (not presently available from APDA) contains the most up-to-date and correct parameter lists for the audio calls. •

DStatus AND DControl

Once you've found the drive, exchanging information with it is simply a matter of **`DStatus`** and **`DControl`** calls. **`DStatus`** enables you to receive status data from the drive; **`DControl`** enables you to send control data to the drive.

The main parameter table for **`DControl`** and **`DStatus`** contains a parameter count, the device number you're working with, the control (or status) code, a pointer to the command data, a request count (used for status calls), and a transfer count.

The command data information is a parameter list of 18 bytes (see Figure 1). The first two are reserved and must be 0; the following byte is the SCSI command (which is the same as the control/status code low byte). Next is a block of 11 bytes: these are specific to each call. Finally, the command data parameter list ends with a long pointer to another buffer, where SCSI data is returned from the status calls.

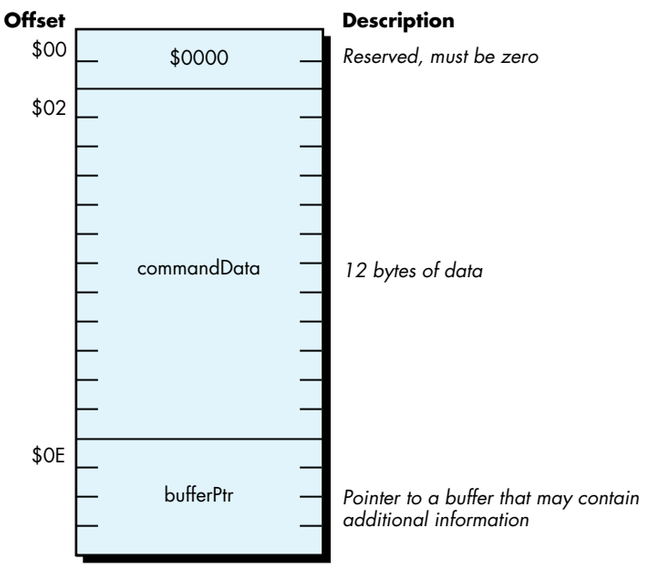


Figure 1
Command Data for DStatus and DControl

The following code from the sample program implements two handlers to make **DControl** and **DStatus** calls:

```
* Make a DControl call - enter with control code in accumulator.

DoDControl
      sta   DCcode     ;Store control code.
      shortacc
      sta   controlData ;Store it in start of the parameter list.
      longacc

      jsl   GSOS
      dw    DControl   ;Make GS/OS DControl call.
      adrl  :devParm

      jsr   GDS        ;Get device status & set new disc flag, if necessary . . .

      rts              ;and return with the call made.

:devParm dw    5       ;Parm list for the DControl call.
DCdevNum dw    0       ;Fill in device number here.
DCcode   dw    0       ;Control code.
      adrl  controlList ;Pointer to buffer.
      dl    0          ;RequestCount - unused.
      dl    0          ;TransferCount.

controlList dw 0       ;Reserved.
controlData ds 12      ;12 bytes of data.
      adrl  buffer     ;Pointer to buffer.
buffer   ds    20

* Make a DStatus call - enter with status code in accumulator.

DoDStatus
      sta   DScode     ;Store control code.
      shortacc
      sta   statusData ;Store it in start of the parameter list.
      longacc

      jsl   GSOS
      dw    DStatus    ;Make GS/OS DStatus call.
      adrl  statParm

      jsr   GDS        ;Get device status & set new disc flag, if necessary . . .

      rts              ;and return with the call made.
```

```
statParm dw    5        ;Parm list for the DStatus call.
DSdevNum dw    0        ;Fill in device number here.
DScode   dw    0        ;Status code.
      adrl  statusList  ;Pointer to buffer.
DSrequest dl   0        ;RequestCount.
      dl    0           ;TransferCount.

statusList dw  0        ;Reserved.
statusData ds  12       ;12 bytes of data.
      adrl  buffer      ;Pointer to buffer.
```

TO FIND OUT MORE ABOUT THE DISC IN THE DRIVE

The AppleCD SC has two important calls for finding out more about the disc in the drive: **ReadTOC** (to read the table of contents) and **AudioStatus** (to find out exactly what the drive is doing). These calls are useful immediately after the user has inserted a new (foreign) disc, to ascertain the disc's layout and whether the disc is currently playing, searching, paused, or muted. They are also useful in the case where you've placed one application on several different discs, each with a different audio track layout.

READTOC

The **ReadTOC** call can return data in three ways, known as types $00, $01, and $02. You specify the type in the **ReadTOC** parameter list.

A type $00 table of contents returns the value of the first and last tracks available on the disc. Tracks are numbered consecutively, starting with 1, and the type $00 table of contents always returns $01 as the value for the first track on the disc.

Type $01 gives you the disc lead-out time in minutes, seconds, and blocks. (The lead-out time is the total time of all tracks on the disc, including the data track, if one exists.)

Finally, type $02, the most flexible of the three, returns starting address information (control field, minutes, seconds, and blocks) for each track on the disc. You can specify how many bytes the call will return and which track to start on, which makes it possible to find out about a single track instead of all of the tracks on the disc.

Examples of using the **ReadTOC** call can be found in the "NewDisc" and "Play" routines of the accompanying source code.

ABOUT CD-ROM AUDIO TRACK FORMAT

Audio data on a CD-ROM is stored in tracks. A CD can have a maximum of 99 tracks. Each track is broken down into minutes, seconds, blocks, frames, and finally, bytes, as shown in Figure 2. Tracks are numbered consecutively starting with 1, while minutes, seconds, blocks, and frames are numbered consecutively starting with 0.

This format enables exact specification of a location on a CD-ROM. A location can be specified by absolute block number (for example, start playing at absolute block 1,234,567 from the start of the disc), or by absolute minute, second, and block number (for example, start playing at minute 42, second 30, block 15 on the disc), or by logical track number (for example, start playing at track 2).

Note that blocks are often referred to as frames in CD-ROM industry documentation; following that lead, you'll see references to disc data in minute-second-frame format in my source code when it's truly in minute-second-block format. There is no way to access individual frames (1/98th of a block) or bytes of the disc with the AppleCD SC drive; however, for audio playback, it is unnecessary and presents no handicap.

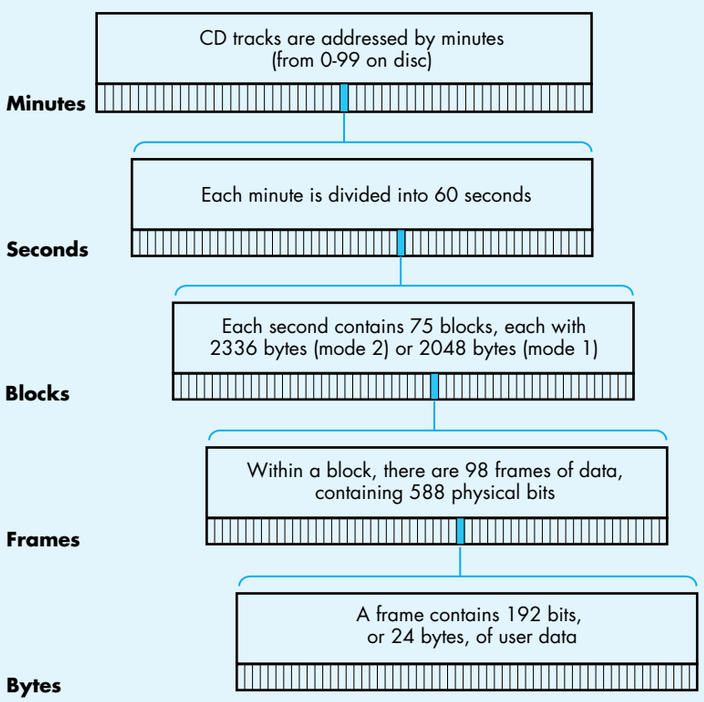


Figure 2
Format of a CD-ROM Audio Track

AudioStatus

The **`AudioStatus`** call returns the current status of the drive, the current play mode, the control field of the current track, and the Q Subcode for either the next track on the disc (if a track is currently playing) or the current track (if a track is not currently playing).

The current status of the drive is reported as one of six messages: AudioPlay operation in progress, Pause operation in progress, Muting On operation in progress, AudioPlay completion status, Error during AudioPlay operation status, or AudioPlay operation not requested.

The play mode is how audio will be output. It has the following possible values:

Bits 3210	Effect
0000	Muting on (no audio)
0001	Right channel through right channel only
0010	Left channel through right channel only
0011	Both channels through right channel only
0100	Right channel through left channel only
0101	Right channel through left and right channel
0110	Right channel through left channel, left channel through right channel (reversed)
0111	Right channel through left channel, both channels through right channel
1000	Left channel through left channel only
1001	Left channel through left channel, right channel through right channel (stereo)
1010	Left channel through left and right channels
1011	Left channel through left channel, both channels through right channel
1100	Both channels through left channel
1101	Both channels through left channel, right channel through right channel
1110	Both channels through left channel, left channel through right channel
1111	Both channels through left channel, both channels through right channel (mono)

The control field describes the format of the current track, and has the following possible values:

Bits 3210	Effect
00x0	Two audio channels without preemphasis
00x1	Two audio channels with preemphasis
10x0	Four audio channels without preemphasis
10x1	Four audio channels with preemphasis
01x0	Data track
01x1	Reserved
11xx	Reserved
xx0x	Digital copy prohibited
xx1x	Digital copy permitted

The Q Subcode is the absolute address of either the next track on the disc (if a track is currently playing) or the current track (if a track is not currently playing). It consists of the track starting address in minutes, seconds, and blocks. With the Q Subcode, you can quickly tell where the laser is positioned.

Here is a short example of getting the drive status and reporting it to the user:

```
Status
         ~WriteCString #:stat

         jsr   ZeroParamList ;Zap old parameter list.

         lda   #$0006        ;Get six bytes from AudioStatus.
         sta   DSrequest
         lda   #AudioStatus  ;Make this call.
         jsr   DoDStatus
         shortacc
         lda   buffer        ;Get audio status.
         longacc
         cmp   #5+1
         bge   :bad
         asl                 ;*2 so offset into table is correct.
         tay
         lda   #^:msgPtrs    ;Get current bank.
         pha                 ;Push high word.
         lda   :msgPtrs,y    ;Push low word.
         pha
         _WriteCString       ;Print string.
         clc
         rts

:bad     ~WriteCString #:unk
         clc
         rts

:stat    asc   'Status'0d0d00
:unk     asc   'Unknown audio status returned'0d00
:msgPtrs
         dw    :nowPlay      ;$00
         dw    :pause        ;$01
         dw    :muting       ;$02
         dw    :playComp     ;$03
         dw    :errPlay      ;$04
         dw    :noPlay       ;$05
```

```
:nowPlay asc   'AudioPlay operation in progress'0d00
:pause   asc   'Pause operation in progress'0d00
:muting  asc   'Muting On operation in progress'0d00
:playComp asc  'AudioPlay completion status'0d00
:errPlay asc   'Error occurred during AudioPlay operation'0d00
:noPlay  asc   'AudioPlay operation not requested'0d00
```

TO PLAY AUDIO TRACKS

Five main audio calls are available to the programmer to control the audio features of the AppleCD SC drive:

AudioPlay This call enables you to start the drive on an audio playback operation (you pass it the play mode), or to specify a stop address for audio playback.

AudioStop Like **AudioPlay**, this call enables you to specify a stop address for audio playback. **AudioStop** can be used to set up a stop address prior to issuing an **AudioPlay** call starting playback.

AudioPause This call enables you to temporarily stop the audio playback operation by turning on muting and holding the laser over the same Q Subcode address. **AudioPause** also enables you to resume the audio playback operation after it has been stopped with a previous **AudioPause** operation. This call is useful if you wish to pause audio playback on the fly and resume it instantly, without any delay.

AudioSearch This call enables you to position the laser over an address on the disc (a specified track or Q Subcode). This can be useful if it is crucial that your application be able to start playback at a certain time: you can first search to the specific track and then hold the disc there, later issuing an **AudioPlay** command (which will begin play immediately). **AudioSearch** can also be set to start playing as soon as the specified address is located.

AudioScan This call causes a fast-forward or fast-reverse scan operation, starting from the address passed to it.

The steps to play an audio track from your application are fairly straightforward:

1. Choose the last track you wish to play and set it with the **AudioStop** command.
2. Choose the first track you wish to play and pass it to **AudioPlay** with the stop flag set to 0.

If you wish to play a single track, pass the same track number for both commands. If you want to simply start playback on a given track and allow the disc to play to the end, pass the last track number to **AudioStop** and the track to begin playback on to **AudioPlay.** (The last track number can be retrieved with a type $00 **ReadTOC** command.)

The **AudioScan** call is useful if you're allowing the user to directly control the disc from the application; otherwise, there is little need to fast-forward or fast-rewind when your program is already aware of the layout of the disc. **AudioPause** and **AudioSearch** are two practical ways to prepare the disc for playback without delay, and then pause and resume it. By using **AudioSearch** before your program needs to begin playback, you can have the disc spinning and the laser positioned exactly where it needs to be.

TO SUM IT ALL UP

By now, you're familiar with the many advantages and possibilities that CD-ROM audio access can provide your application. You know the layout of an audio track on CD-ROM and how to find out more about the track from your application.

You know the five major audio calls and a handful of supporting calls. You know what they can do for you and why you might want to use them.

You're ready to produce an application that takes full advantage of the AppleCD SC drive, equipping your program with the ability to produce sound and music of unequaled quality.

Thanks to Our Technical Reviewers:
Mike Barnick, James Beninghaus, Matt Gulick, Jim Luther, Llew Roberts

Eric sends many thanks to Ken Kashmarek for his help.

SURF'S UP: CATCH THE COMM TOOLBOX WAVE

The Macintosh Communications Toolbox provides managers and utilities that offer basic networking and communications services to applications. This article introduces you to three of the Communications Toolbox managers—the Connection Manager, the Terminal Manager, and the File Transfer Manager—as well as Surfer, a sample application that uses the Communications Toolbox to implement simple networking and communications services.

ROB BERKOWITZ AND ALEX KAZIM

Networking and communications applications running on the Macintosh are like a good pair of rose-colored shades. They filter out the harshness of antiquated architectures and conventions, and present users with a familiar, intuitive interface. The Macintosh Communications Toolbox provides a standard framework in which you can develop modular, consistent networking and communications applications. As a developer using the Communications Toolbox, you can write applications without having to know the complexities of each networking and communications environment your applications run in. For example, imagine writing a chess program that enables users to play opponents over any sort of data connection, without having to code for each type of connection. The Communications Toolbox makes this possible.

COMMUNICATIONS TOOLBOX CONTENTS

The Communications Toolbox consists of four managers and a set of utilities that provide basic networking and communications services. Think of these managers and utilities as an extension to the Macintosh Toolbox. Each of the managers in the Communications Toolbox—the Connection Manager, the Terminal Manager, the File Transfer Manager, and the Communications Resource Manager—handles a different aspect of networking and communications. The utilities provide routines that perform a variety of useful auxiliary functions. This article focuses on the Connection Manager, the Terminal Manager, and the File Transfer Manager.

ROB BERKOWITZ'S career has come a long way, despite getting off to a dubious start (he worked for the large blue corporation). His B.S. in English (emphasis on the BS, he says) from Carnegie Mellon put him on the path to his Great American Reference Manual, the *Macintosh Toolbox Reference*. He likes working at Apple because "the offices are right next to some pretty primo cycling trails" and he has "the freedom to thrash around in the dirt in the middle of the day all year round." He is a Grateful Dead enthusiast who feels that "most Deadheads are genuinely good people. It would be nice if more people were like that." Truck on, Rob.

Communications Toolbox managers work with communications tools, which are self-contained software modules that provide protocol-specific services. The managers and tools perform the following functions:

- The Connection Manager and tools are the mechanism for establishing and maintaining a data connection between machines.
- The Terminal Manager and tools show data to users in a manner that emulates the characteristics of specific terminal types.
- The File Transfer Manager and tools handle the protocols for sending and receiving files.
- The Communications Resource Manager helps applications keep track of necessary resources.

You code to the application programming interface defined by the managers. In turn, the managers request specific services from communications tools. The interaction between the tool and the manager is invisible to your application, so when you design your application, you don't have to be concerned with what sort of data connection is in place, what kind of terminal to emulate, or what type of file transfer to perform. It's similar to the way applications deal with the Printing Manager. Applications say "Print," and the Printing Manager sends the request to the Printer Driver, which figures out how to print on a specific device.

By providing basic services, communications tools free application developers from having to learn the most heinous intricacies of communications conventions. That work is left up to those who practice the black art of writing communications tools.

Communications tools live in the Communications folder, which is in the System Folder and is created and populated when you install the Communications Toolbox. (Under System 7.0 they will reside in the Extensions folder.) A number of communications tools are available from APDA. Others, from Apple and third-party developers, will be available in the near future.

Figure 1 shows how the Communications Toolbox managers and tools fit between your application and the operating system. The application interacts with the manager, which in turn interacts with the tool. The tool, in turn, communicates with the operating system (or Communications Resource Manager), provides a specific service, and passes back to the application (through the manager) any relevant informationp.

KAZ doesn't know his official title, but thinks he may be a Communications Toolbox Engineer. We try to be understanding of these lapses; he's got his hands full keeping track of and remembering the names of his hundreds of colorful family members worldwide. He, himself, is an international sort; he was born in Trinidad and has lived in Toronto and Texas. He's been at Apple since 1988, after getting his degree in mechanical engineering from Rice University. He tries to stay as busy as possible, especially with sailing, skiing, gliding, cooking, dancing, and writing fiction. He likes his food extremely(!) spicy, so we suggest caution if he invites you to lunch. His goal in life is not to get convicted, and he studies karate. Again, we suggest caution if he invites you to lunch.

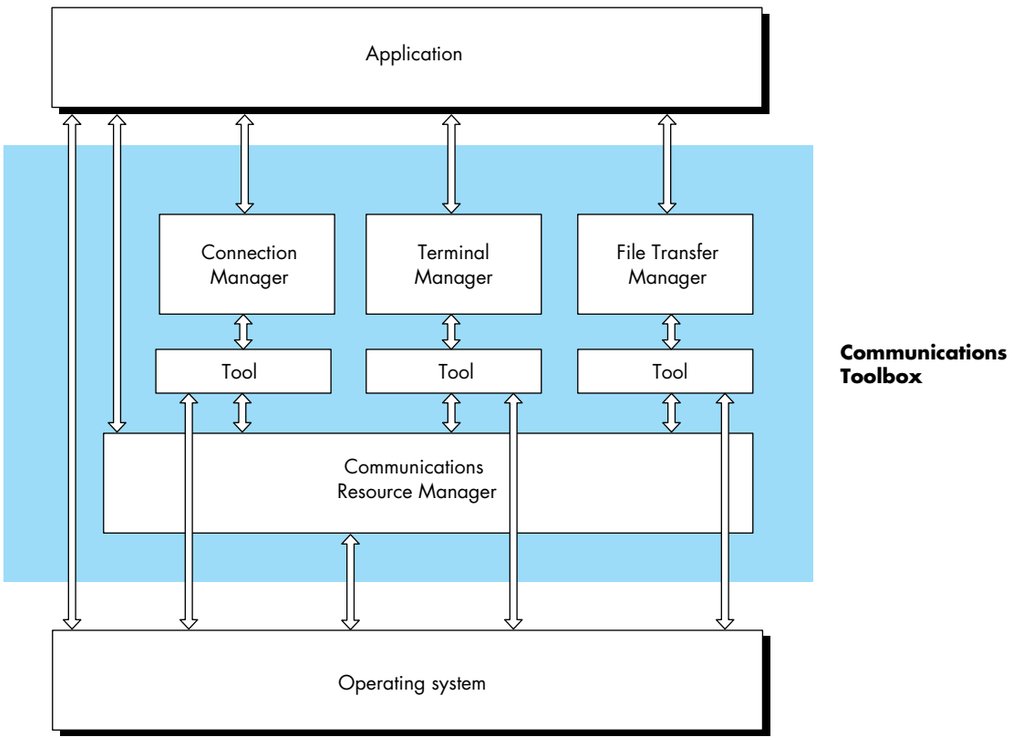


Figure 1
How the Communications Toolbox Fits In

USING THE MANAGERS: AN OVERVIEW

In this section we give an overview of how your application uses each manager. The sample application Surfer, discussed in detail in the next section, provides a model of how an application uses the Communications Toolbox.

To get each of the Communications Toolbox managers ready for action, your application does the following:

1. Initializes the manager, by calling **`InitCM`**, **`InitTM`**, or **`InitFT`**.
2. Gets the **`procID`** for a specified tool (this is a tool file reference number, similar to the ones returned by the File and Resource Managers), by calling **`CMGetProcID`**, **`TMGetProcID`**, or **`FTProcID`**.

3. Creates a new instance of the tool, by calling **CMNew**, **TMNew**, or **FTNew**.
4. Configures the tool. You can present users with a standard tool-settings dialog box by calling **CMChoose**, **TMChoose**, or **FTChoose**, or you can set the configuration directly using a configuration string. Figure 2 shows the dialog box that is put up in response to a call to **TMChoose** while the VT 102 tool is active.

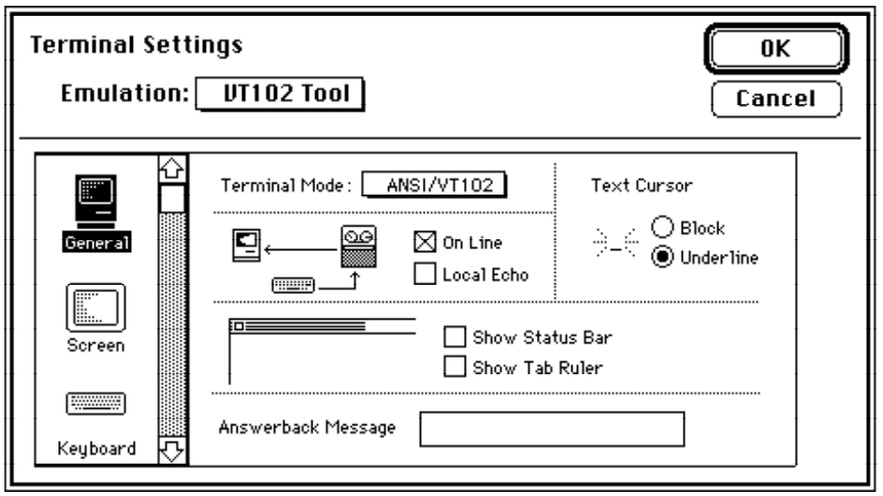


Figure 2
The Dialog Box for Choosing and Configuring a Terminal Tool

Associated with each communications service (connection, terminal, and file transfer) is a data structure that the manager, tool, and application maintain. For the Connection Manager, this data structure is called the connection record; for the Terminal Manager, the terminal record; for the File Transfer Manager, the file transfer record. These records are discussed in detail in *Macintosh Communications Toolbox Reference*. Your application refers to these records for information upon which to base decisions, similarly to how the Window Manager uses information in the window record.

An important concept central to the architecture of the Communications Toolbox is that applications wait on events. When an application (Surfer, for example) gets an event to pass to one of the communications tools, it tells the manager, which then passes a message to the appropriate tool, along with a handle to the associated data structure (that is, connection record, terminal record, or file transfer record). The communication between the tool and the manager is done through the data structure and return codes. The section Handling Events goes into more detail about this.

The definitive reference on the Communications Toolbox is *Macintosh Communications Toolbox Reference*, available from APDA.

A LOOK AT SURFER

Surfer is a simple terminal emulation package that Alex adapted from the DTS sample code sent out to developers. It uses the Communications Toolbox to implement simple networking and communications services. It provides support for data connections, terminal emulations, and file transfers; and can use new communications tools without changing one line of code. Keep in mind that you can use the Communications Toolbox to go well beyond the domain of standard terminal emulation software, to seamlessly incorporate networking and communications functionality into all kinds of programs.

Here we'll show you selected portions of Surfer to illustrate how it uses the Communications Toolbox. First we'll show you code to help you get a feel for the structure and flow of the program. Then we'll show how Surfer meets the common communications challenges of establishing and maintaining a connection, emulating a terminal, and tranferring files. Finally, we'll discuss how Surfer handles two common problems. You should examine the source code, which appears in its entirety on the *Developer Essentials* disc, to fully understand Surfer. You should also examine the connection record, terminal record, and file transfer record in Surfer. As mentioned earlier, these records are fundamental to the operation of each manager.

HOW SURFER STARTS UP

Here's Surfer's main routine:

```
BEGIN
      UnloadSeg(@_DataInit); { Note that _DataInit must not be in Main! }
      MaxApplZone;           { Expand the heap so code segments load at the top.
}
      Initialize;            { Initialize the program. }
      UnloadSeg(@Initialize);{ Note that Initialize must not be in Main! }
      EventLoop;             { Call the main event loop. }
END.
```

As with surfing, where you've got to get out the wetsuit and put the board on the Bug, Surfer has some preparation it needs to do before it calls its main event loop. The following fragment from the **Initialize** procedure shows how Surfer initializes the Communications Toolbox:

```
{ Does CommToolbox exist? }
IF NOT TrapAvailable(_CommToolboxTrap, OSTrap) THEN
      AlertUser('ACK!! No CommToolbox',TRUE);

{ Check for System 6.0 or better, 64K ROM. }
ignoreError := SysEnvirons(kSysEnvironsVersion, TerraMac);
```

```
WITH TerraMac DO
      IF (systemVersion < $0600) OR (machineType < 0) THEN
            AlertUser('Need System 6.0 or better',TRUE);

{ Check various memory configs. }
IF ORD(GetApplLimit) - ORD(ApplicZone) < kMinHeap THEN
      AlertUser('Out of Memory',TRUE);

PurgeSpace(total, contig);
IF total < kMinSpace THEN
      AlertUser('Out of Memory',TRUE);

{ Load up the Communications Toolbox. }
{ Must initialize CRM & CTBUtilities first. }
err :=      InitCTBUtilities;
err :=      InitCRM;

err := InitTM;          {Initialize the Terminal Manager.}
IF err = TMNoTools THEN
      AlertUser('No terminal tools found',TRUE);

err :=      InitCM;     { Initialize the Connection Manager. }
IF err = CMNoTools THEN
      AlertUser('No connection tools found',TRUE);

err :=      InitFT;     { Initialize the File Transfer Manager. }
IF err = FTNoTools THEN
      AlertUser('No file transfer tools found',FALSE);
```

HANDLING EVENTS

After dealing with the initialization details, Surfer loops, waiting for events to wave through, handling them like this:

```
PROCEDURE EventLoop;
VAR
    gotEvent     : BOOLEAN;
    event        : EventRecord;

BEGIN
    REPEAT
          DoIdle;

          IF gHasWaitNextEvent THEN { Put us 'asleep' forever under MultiFinder }
                gotEvent := WaitNextEvent(everyEvent, event, 0, NIL)
```

```
ELSE BEGIN
                SystemTask;     { Must be called if using GetNextEvent. }
                gotEvent := GetNextEvent(everyEvent, event);
          END;

          IF gotEvent THEN BEGIN
                AdjustCursor(event.where);{ Make sure we have the right cursor. }
                DoEvent(event);
          END;

          AdjustCursor(event.where);
    UNTIL FALSE;                { Loop forever; we quit through an ExitToShell. }
END; { EventLoop }
```

The procedure **DoEvent** is where much of the code surfing takes place. The procedure is too long to reproduce here, but Figure 3 shows the important points schematically, and you can read the source code on the CD for more details.

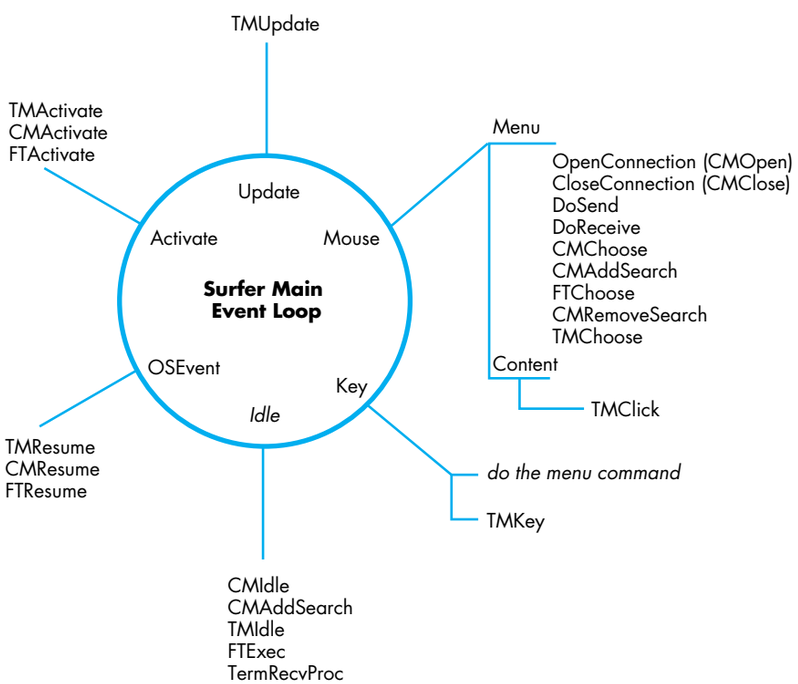


Figure 3
How Surfer Handles Events

Events received by Surfer come in two main flavors: Surfer-owned and tool-owned. Tools can create their own windows behind Surfer's back (for instance, file transfer tools can put up a status window during a transfer), but since Surfer is in control, events destined for these windows come through Surfer's main event loop. Luckily, when a tool creates its own window, it stuffs a handle to itself in the window's **refCon** field. All Surfer has to do to determine who owns a window is compare the window's **refCon** to the existing tool handles. If a match is found, Surfer calls the appropriate routine (**TMEvent**, **CMEvent**, or **FTEvent**) so that the tool can handle the event. Otherwise, Surfer handles the event itself.

Even Surfer's window, though, has "guests." The terminal tool needs to receive mouse clicks and update events, and all the tools need to receive activate/deactivate and suspend/resume events. These are passed to the tool with the appropriate routine: **TMClick**, **TMUpdate**, **CMActivate**, and so on.

MANAGING IDLE TIME

The idle procedure is a little convoluted because so much is happening. Surfer has to read and send data someplace, blink cursors, and provide all the sessions time for background file transfers. It's something like the action at O'Hare Airport.

Because communications tools can display their own windows, Surfer must walk the window list to give each tool time in a session. Although idle time is provided to tools with **CMIdle**, **FTExec**, and **TMIdle**, Surfer doesn't want to send data to the terminal tool if there's a file transfer in progress and the tools are using the same channel. Another thing worth checking is the file transfer status. Did the transfer start? Has it just ended? Was it successful? Did an auto-receive sequence come across? These concerns are discussed in greater detail later in this article in the section "Doing a File Transfer."

Surfer also needs to get data to the terminal tool. Surfer checks to see if the connection is open or data is available, and if so tells the tool to read it and stream it to the terminal:

```
{ Get the state of the connection. }
theErr := CMStatus(gConn, sizes, status);

IF (theErr = noErr) THEN  BEGIN
      { Route the data if we have any. }
      IF (BAND(status, cmStatusOpen + cmStatusDataAvail) <>  0) AND
      (sizes[cmDataIn] <>  0) THEN BEGIN

            { Tell the tool to get the data. }
            theErr := CMRead(gConn, gBuffer, sizes[cmDataIn],
                        cmData, FALSE,NIL,0,flags);
```

```
            { Send data to the terminal. }
            IF (theErr = noErr) THEN
                  bytesEaten := TMStream(gTerm,gBuffer,
                                 sizes[cmDataIn],flags);

            { Could check bytesEaten vs. sizes[cmDataIn]. }
      END; { Sizes <>  0. }
END; { Good status. }
```

Now that you've seen something of Surfer's main event loop, we'll look at how Surfer uses the Communications Toolbox managers.

INITIATING A CONNECTION

For two computers to talk to each other, they must establish and maintain a data connection. Applications that provide terminal emulation or file transfer services use the data connection to physically transfer the data. Before an application can open a data connection, the Connection Manager has to be properly set up. Surfer does this during initialization by calling **InitCM** to initialize the Connection Manager; calling **CMGetProcID** to get its **ProcID**; and calling **CMNew** to create a new instance of a connection tool. Note that Surfer does not explicitly configure the tool: **CMNew** automatically configures the tool to its default settings. The user can reconfigure the tool by choosing the appropriate menu item.

Here's how Surfer calls **CMNew**:

```
sizes[cmDataIn] := kBufferSize;
sizes[cmDataOut] := kBufferSize;
sizes[cmCntlIn] := 0;
sizes[cmCntlOut] := 0;
sizes[cmAttnIn] := 0;
sizes[cmAttnOut] := 0;

{ refCon and UserData are 0. }
gConn := CMNew(procID, cmData, sizes, 0, 0);
IF gConn = NIL THEN
      AlertUser('Can''t create a connection tool',TRUE);
```

Surfer supports only the data channel and asks for a 1K buffer for both input and output. Because the connection tool may be unable to handle the requested buffer size, Surfer needs to look at the **bufSizes** field in the connection record and use that value to allocate space for the buffer.

```
gBuffer := NewPtr(gConn^^.bufSizes[cmDataIn]);
IF MemError <> noErr THEN
      AlertUser('Out of memory',TRUE);
```

Before Surfer initiates a connection, it checks the state of the connection with **CMStatus**. If the connection is not already open or in the process of opening, Surfer issues a **CMOpen** call, in this case synchronously with a 0 time-out. A timeout of 0 says, "Make a single attempt to open the connection."

```
{ Get connection info.        }
theErr := CMStatus(gConn, sizes, status);
IF (theErr = noErr) THEN BEGIN
      { If it isn't already open, then open it. }
      IF BAND(status, cmStatusOpen + cmStatusOpening) = 0 THEN
            theErr := CMOpen(gConn, FALSE, NIL, 0);
END;
```

ONCE THE CONNECTION IS OPEN . . .

Using the Connection Manager to maintain a data connection is a lot like talking with a friend. You start off by saying hello (**CMOpen**, **CMAccept**, or **CMListen**), ask how he's doing (**CMStatus**), engage in small talk (**CMRead** and **CMWrite**), take a deep breath once in a while (**CMIdle**), and end by saying goodbye (**CMClose**).

Surfer uses **CMStatus** a lot to return information like whether there is data waiting to be read, and whether the connection is open or closed. If **CMStatus** shows that data is available, Surfer reads it and passes the data to the terminal tool or the file transfer tool, whichever is appropriate.

Closing a connection is similar to opening one, except, of course, the logic is reversed: Surfer only closes the connection if it's open. When Surfer is done with the session, it disposes of the tool with **CMDispose** and gets rid of the buffer with **DisposPtr**.

STARTING A TERMINAL EMULATION SESSION

In Surfer, a session is a connection, terminal, and file transfer tool, along with a data buffer tied to an owning window. Surfer is limited to one window and one session. To create a new session, Surfer calls

```
{ Get window. }
window :=  GetNewWindow(rWindow, NIL, WindowPtr(-1));
SetPort(window);
```

The first thing to notice right after the **GetNewWindow** call is the **SetPort**. The terminal tool does as little port manipulation as possible and assumes that the port where it's drawing is the correct one. Surfer sets the port so it can direct drawing to either a window graphics port or a printer graphics port. Be warned that the terminal tool may die ungracefully if the current port is not valid.

During initialization, Surfer gets the Terminal Manager ready for action by calling **InitTM**, **TMGetProcID**, and **TMNew**. Surfer creates a new instance of a terminal tool as follows:

```
theRect := window^.portRect;

{ Flags set to 0; no cacheProc, breakProc, or clikLoop;     }
{ refCon and UserData are 0.                                }
gTerm := TMNew(theRect,theRect,0,procID,window,
              @TermSendProc,NIL,NIL,NIL,@ToolGetConnEnvirons,0,0);
IF gTerm = NIL THEN
      AlertUser('Can''t create a terminal tool',TRUE);
```

In **TMNew**, Surfer specifies the bounds of the drawing area, the terminal tool to use, the owning window, two internal procedures, and some application data.

When Surfer is done with the session, it disposes of the tools with **TMDispose**.

ONCE THE TERMINAL SESSION HAS STARTED . . .

The Terminal Manager handles the interaction between the host and the user during a session. Through the Terminal Manager, terminal tools can display both words and images to the user in a manner that emulates the characteristics of specific terminal types. The Terminal Manager also accepts information from the user (such as keystrokes), which is sent back to the host.

Many Terminal Manager routines are similar to TextEdit routines. Since the Terminal Manager is in charge of interacting with the user, most of the calls to the Terminal Manager that Surfer uses are event-oriented—even events like streaming data, when data becomes available, and passing data between Surfer and the tool.

The calls **TMIdle**, **TMStream**, and **TMKey** enable Surfer to provide basic terminal emulation services. Surfer calls **TMIdle** during its idle loop so, among other things, the tool can blink its cursor (similar to the way **TEIdle** works). When data becomes available from the connection tool, Surfer calls **TMStream** to stream the data to the terminal tool for drawing to the window. When a key event occurs, Surfer calls **TMKey**. The terminal tool processes the keystroke and then uses a Surfer procedure, specified in **TMNew**, to send data back to the connection. Since Surfer is in charge of this procedure, it can do some data filtering, use synchronous or asynchronous write calls, or just drop the request on the floor if it wants to.

Your application will probably support multiple sessions, making it difficult to find the connection handle associated with a particular terminal record. To help your application out, the terminal tool's **refCon** is passed along in the parameter list to **TMNew**. Your application can put the connection tool handle in this location for

this purpose. Then, in **TermSendProc** (a procedure that terminal tools expect your application to provide), your application can get the connection handle back by casting the **refCon** to a **ConnHandle**. Of course, this is only one possibility; you can store whatever you want in the **refCon**.

DOING A FILE TRANSFER

The File Transfer Manager provides file transfer services for a transfer between Surfer and another computer process. The other process can be running on the same computer as Surfer or on any other type of computer. Surfer makes a request of the File Transfer Manager in order to transfer a file or perform some other file transfer related function. The File Transfer Manager then sends this request to one of the tools it manages. The tool provides the service according to the specifics of its file transfer protocol. Once the tool has finished, it passes back to Surfer any relevant parameters and return codes. It's very similar to the way the other Communications Toolbox managers work.

During initialization, Surfer gets the File Transfer Manager ready for action by calling **InitFT**, **FTGetProcID**, and **FTNew**. Surfer uses the following code to create a new instance of a file transfer tool:

```
{ Flags set to 0, no read/write proc (let the tool use its own),  }
{ refCon and UserData are 0.                                      }
gFT := FTNew(procID,0,@FTsendProc,@FTreceiveProc,NIL,NIL,
            @ToolGetConnEnvirons,window,0,0);
IF gFT = NIL THEN
       AlertUser('Can''t create a file transfer tool',TRUE);
```

For **FTNew**, Surfer specifies a send-and-receive procedure for the file transfer tool to use, if it doesn't already have one. Some file transfer tools, like ftp, handle their own connection and therefore don't use these procedures.

To start a file transfer, either sending or receiving, Surfer calls **FTStart**. To keep the transfer going, Surfer calls **FTExec** in its idle loop. That's it. When the transfer has completed, the tool takes care of closing itself. If Surfer needs to stop during the transfer, it can call **FTAbort**, and the tool automatically cleans up its mess.

Surfer needs to handle three things during a file transfer. First, it needs to look out for an auto-receive string, a sequence of characters supported by some file transfer protocols that throws Surfer into receive mode (MacTerminal 1.1 does this). If the file transfer tool supports auto-receive strings, Surfer uses the Connection Manager routine **CMAddSearch** to tell the connection tool to look out for the auto-receive string. Incidentally, when the connection tool is looking for an auto-receive string and the user chooses a new connection tool or modifies the current one, the Connection Manager destroys the old search for this string. Surfer, therefore, needs to add the search again.

Second, Surfer needs to handle data routing. Most file transfer tools use the current connection to get data. However, if a file transfer is in progress, we don't want Surfer trying to send data to the terminal tool. Some file transfer tools establish their own connection separate from the one Surfer has established, so any data read from the connection should go to the terminal tool as usual.

Third, Surfer needs to check that the file transfer was copacetic. Here's how it does this. During a file transfer, the File Transfer Manager turns on a bit in the file transfer record called **ftIsFTMode**. By keeping track of this bit, Surfer can tell when a file transfer has completed. It can then check the **FTSucc** bit in the file transfer record to see if the file transfer went according to plan.

Two of the procedures file transfer tools use are **FTSendProc** and **FTReceiveProc**, which respectively send and receive data. **FTSendProc** and **FTReceiveProc** are similar to **TermSendProc**, except the file transfer tool can specify which connection channel Surfer should use to read or write the data.

When Surfer is done with the session, it disposes of the tools with **FTDispose**.

HOW SURFER WORKS WITH AUTO-RECEIVE STRINGS

Whenever a new file transfer tool is created, either through an **FTNew** or **FTChoose**, Surfer searches the file transfer record for an auto-receive string. If there is one, Surfer calls **CMAddSearch** to tell the Connection Manager to look for the string in the incoming data.

```
IF (gFT <> NIL) AND (gConn <> NIL) THEN BEGIN
      tempStr := gFT^^.AutoRec;     { Do I need to add a search?        }
      IF (tempStr <> '') THEN BEGIN
            gFTSearchRefNum := CMAddSearch(gConn,tempStr,cmSearchSevenBit,
                                                   @AutoRecCallback);
            IF gFTSearchRefNum = -1 THEN BEGIN
                  AlertUser('Couldn''t add stream search',FALSE);
                  gFTSearchRefNum := 0;
            END;
      END; { Can autoreceive. }
END; { Good FT and conn.      }
```

Surfer passes a **procPtr** to **CMAddSearch** so that when the search completes, the connection tool calls Surfer's **AutoRecCallback**. If more than one search was going on simultaneously, Surfer also gets back a **refNum** to help identify the returning search.

When the file transfer tool calls **AutoRecCallback**, Surfer starts to receive a file. Unfortunately, Surfer can't call **FTStart** from the callback procedure, because that procedure may be called at interrupt time, and **FTStart** cannot be called at interrupt time because it may move memory. So Surfer does the next best

thing. It sets a global flag in **AutoRecCallback** that says it received the auto-receive string. During the idle loop, it then looks at the flag to see if it's time to start the file transfer.

Here's how Surfer start to receive a file transfer.

```
IF gFT <> NIL THEN BEGIN
      { Let the FT tool use its own default file info.      }
      theReply.vRefNum := 0;
      theReply.fName := '';
      theReply.good := TRUE;

      gStartFT := FALSE;            { Shut the flag down.   }

      { We remove the search temporarily in case it comes   }
      { across during the transfer. Will be re-added in the }
      { idle loop once the transfer is completed.           }

      IF gConn <> NIL THEN
            IF (gFT^^.autoRec <> '') AND (gFTSearchRefNum <> 0) THEN BEGIN
                  CMRemoveSearch(gConn, gFTSearchRefNum);
                  gFTSearchRefNum := 0;   { We found it already.  }
            END;

      { Start receiving the file.                     }
      { The rest gets transferred in the Idle loop.   }

      anyErr := FTStart(gFT,ftReceiving,theReply);

      IF (anyErr <> noErr) THEN
            ;     { File Transfer tool will alert user on an error.     }
END; { Good handle.     }
```

One other thing to think about is the string itself. If the string randomly came across again during the file transfer, Surfer doesn't want to start the transfer again because it's already in progress. Therefore, when starting the transfer, Surfer removes the search for the string, transfers the file, and adds the search back in the idle loop when finished.

HOW SURFER HANDLES TWO COMMON PROBLEMS

Two useful routines—**IsAppWindow** and **FindToolID**—help Surfer determine the owner of a window and the **procID** for a given tool.

As discussed earlier, a tool-owned window has a handle to the owning tool in its **refCon.** In **IsAppWindow**, Surfer makes sure the window is an application window by checking the **refCons** against all the tool handles.

```
IF window = NIL THEN
      IsAppWindow := FALSE

ELSE BEGIN
      theRefCon := GetWRefCon(window);
      WITH WindowPeek(window)^ DO
            IsAppWindow := ((windowKind > = userKind) |
(windowKind = dialogKind)) &
            (gTerm <> TermHandle(theRefCon)) &
            (gConn <> ConnHandle(theRefCon)) &
            (gFT <> FTHandle(theRefCon));
END;
```

As mentioned earlier, all three managers—the Terminal, Connection, and File Transfer Managers—require a **procID** when specifying a new instance of a tool. To go from a name of a terminal tool, for instance, to a **procID**, Surfer calls **TMGetProcID(theName)** first. (You can also do this with Connection Manager and File Transfer Manager routines.) Because the **procID** is dynamic, Surfer works with the name of the tool, rather than this value.

```
IF (toolClass = ClassTM) THEN BEGIN
      { If it can't get the default, get the first.   }
      toolName := kDefaultTermTool; {VT102 Tool}
      procID := TMGetProcID(toolName);

      IF (procID = -1) THEN BEGIN
            anyErr := CRMGetIndToolName(toolClass,1,toolName);
            IF (anyErr = noErr) THEN
                  procID := TMGetProcID(toolName);
      END;
END { ClassTM. }
```

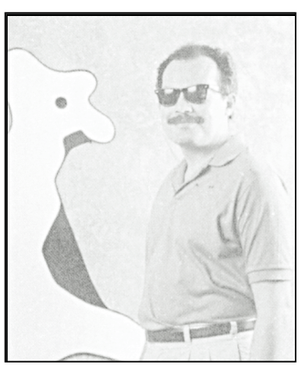
THE END OR JUST THE BEGINNING?

It should be clear by now that the Communications Toolbox makes writing full-blown communications applications and adding networking and communications services to existing applications easier than it used to be. By coding to the Communications Toolbox application programming interface, you can focus on providing networking and communications services rather than worrying about support for various industry standards. Seamless and easy access to information all over the world is revolutionizing how we communicate and think about each other. The Macintosh Communications Toolbox helps application developers and users become part of this revolution.

Thanks to Our Technical Reviewers:
Mark Baumwell, Mary Chan, Byron Han,
Rob Neville, Mike Shoemaker

MACINTOSH DISPLAY CARD 8•24 GC: THE NAKED TRUTH

The Macintosh Display Card 8•24 GC, which supports monitors with depths up to 24 bits per pixel, allows use of a special version of 32-Bit QuickDraw that improves drawing performance on computers in the Macintosh II family. This article details the new card's features, describes how third-party developers can make sure their products are compatible, and tells how to take advantage of the card and its software.



GUILLERMO ORTIZ

Since the invention of the first computer, users and developers have been complaining about the amount of memory available, the speed of the machine, or the number of things that could be done at the same time. (Do you remember when 16K was considered a lot of memory? Of course not.)

The new Macintosh Display Card 8•24 GC addresses the perennial issue of speed, specifically the speed with which the Macintosh executes drawing commands. With its on-board processor and accompanying software, the new 8•24 GC card can execute QuickDraw™ commands from 5 to 30 times faster than standard 32-Bit QuickDraw.

The Macintosh Display Card 8•24 GC has two independent components: the circuitry that controls the display and the hardware and software that accelerate 32-Bit QuickDraw.

DISPLAY NUTS AND BOLTS

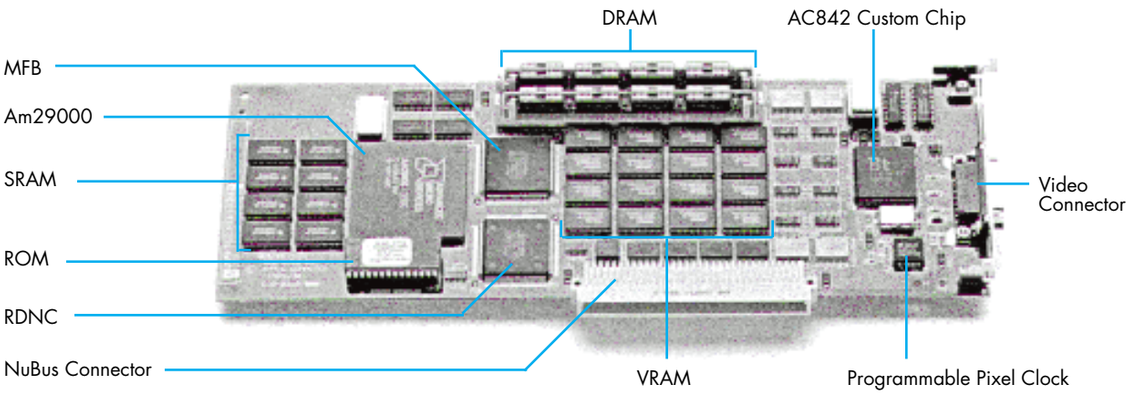
We'll start with the display circuitry of the Macintosh Display Card 8•24 GC, which closely matches the behavior of the Macintosh Display Card 8•24. Figure 1 shows the principal components.

GUILLERMO ORTIZ (in a rare interview): *How long have you been with Apple and why?* Six years plus, and we've got good people and better beer busts.

Did you enjoy your sabbatical? Yes, the energy it restored lasted a full two weeks.

What's the best book you recently read? Between Past and Present by Neil A. Silberman, a new look at archeological findings and discussion of how facts present and past are seen according to the political/social currents of the times.

What're you reading now? Invisible Matter and the Fate of the Universe by Barry Parker. He says that according to current knowledge, 90 percent of the matter of the universe is missing. This leaves some possibilities:



MFB Frame Buffer Controller
Controls the flow of video data and acts as the memory management unit for the Am29000.

Am29000
RISC-architecture processor running at 30 MHz. For those with mainframe mentality, Am29000 performance is measured at 22 MIPS.

SRAM
64K of fast static RAM used to cache Am29000 instructions.

Configuration ROM
Holds the declaration structures, sResources, and so forth; carries the initialization software (Primary and Secondary Inits) and holds the card's driver.

RDNC NuBus Controller
Controls the traffic to and from the NuBus bus. Allows for block transfers both as master and slave card.

DRAM
Up to 8 MB of dynamic RAM used by the GC software.

AC842 Custom Chip
Provides video support; has three digital-to-analog converters and three dual-port color tables capable of supporting 256 8-bit levels each.

VRAM
2 MB of video RAM for buffering the display.

Programmable Pixel Clock
At boot time the card detects the monitor being used and programs this chip to generate the signal that defines the period for refreshing each individual pixel.

Figure 1
Macintosh Display Card 8•24 GC

MONITOR SUPPORT

The new card supports all Apple monitors, including the Apple 13-Inch (B/W & Color) monitor, the Apple Portrait Display, and the Apple Two-Page Display.

The Macintosh Display Card 8•24 GC connects to a monitor through a DB-15 connector and, via the sense lines, detects the type of monitor to which it is connected. At boot time the card senses the monitor and configures itself for it. Third party-monitors with sense lines that are compatible with Apple standards should work as well.

a. We don't have a clue of what is going on.
b. Our current models don't closely enough represent the universe.
c. Physical laws don't apply out of the local universe.
d. I don't know what I am talking about.
e. All or none of the above.

Now a paid political announcement: "Contrary to the lies of his opponents, Mark Harlan is *not* a crook. Trust me!" •

One important difference between the 8•24 GC card and its companion Macintosh Display Card 8•24 is that the 8•24 GC does not fall back into a dormant state when it finds no monitor connected during boot time. To allow for cases when users may want to have an 8•24 GC to accelerate output to other monitors, but don't want to connect a monitor to the card, the 8•24 GC card remains active, allowing 8•24 GC software to operate.

DEPTH SUPPORT

The 8•24 GC card supports depths of up to 8 bits for all monitors and 24-bit color for the 13-inch monitor (and RS-170A and PAL monitors). The 8-bit-per-pixel support for all Apple monitors implies, among other things, that true gray-scale output is possible with both the Apple Portrait Display and the Apple Two-Page Display.

On a high-resolution (13-inch) monitor, the 8•24 GC card can display 24 bits of color per pixel. As many people have noted (repeatedly), with this level of depth support, applications can display more colors than most human eyes can discern.

NUBUS BLOCK TRANSFERS

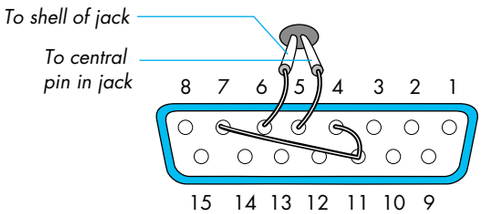
The new card uses NuBus™ block transfers to accelerate display in other display buffers present in the system. The Macintosh Display Card 8•24 GC is a NuBus master card. When it detects other cards in the system that can accept block transfer calls, the card uses block transfer mode to improve the performance of QuickDraw operations all across the board. For video buffer cards that do not support block transfer mode, the Macintosh Display Card 8•24 GC uses a special pseudo block transfer to optimize video performance.

Hardware developers should take special interest in designing cards that will make use of the block transfer mode, which will allow them to squeeze the most performance out of NuBus. The 8•24 GC card can receive block transfers (acting as slave), so other cards that may move data to the card's memory can move the data faster.

This is probably a good place to emphasize that, although you can have as many 8•24 GC boards as you want (or as many as you can afford), only one will function as a graphics accelerator. At boot time QuickDraw GC uses the card in the lowest slot. Any other 8•24 GC card becomes a glorified display card that is basically equivalent to the Macintosh Display Card 8•24.

VIDEO SIGNALS

The RS-170A standard video signals produced by the 8•24 GC card allow you to connect computers in the Macintosh II family to NTSC devices. Note, however, that the RS-170A signals are not directly NTSC compatible. A box is necessary to produce NTSC output that can be displayed on a standard television set or used along with other video equipment such as a VCR. You can generate NTSC black-and-white output by making a cable that uses the green video signal (which also carries a sync pulse), as shown in Figure 2.



Pins 4 and 7 (SENSE0 and SENSE1) are connected to ground, and pin 10 (SENSE2) is not connected at all. This arrangement signals the card to go into the RS-170A mode. When in this mode, pin 5 has the green video plus the sync pulse, and pin 6 carries ground.

Pin no.	Signal name	Pin no.	Signal name
1	Red ground	9	Blue video
2	Red video	10	SENSE2
3	/CSYNC	11	Ground
4	SENSE0	12	/VSYNC
5	Green video	13	Blue ground
6	Green ground	14	Ground
7	SENSE1	15	/HSYNC
8	Not connected		

A slash (/) at the beginning of a signal name indicates that the signal is active low.

SENSE0	SENSE1	SENSE2	Monitor
Ground	Ground	Ground	Reserved
Not connected	Ground	Ground	Apple Portrait Display
Ground	Not connected	Ground	Reserved
Not connected	Not connected	Ground	Apple Two-Page Display
Ground	Ground	Not connected	RS-170A Monitor
Not connected	Ground	Not connected	Reserved
Ground	Not connected	Not connected	High-Resolution 13-Inch Monitor, B&W or Color
Not connected	Not connected	Not connected	No display connected or extended protocol

Figure 2
Simple NTSC B/W Connector

When displaying images Macintosh monitors refresh all the scan lines every time they refresh the screen. This process is called noninterlaced video. RS-170A monitors, which are interlaced devices, can scan only half the lines during each vertical scan. Every other line is “repainted” every time the screen is refreshed.

Interlaced video is reasonably good for pictures and images in general, but really poor for the display of thin horizontal lines, which seem to flicker when repainted only every other scan. Because the Macintosh desktop is ordinarily full of such lines, the desktop looks bad when displayed on interlaced devices. To overcome this problem, the RS-170A output of the 8•24 GC card uses a technique called Apple convolution. The technique is basically a filter that, before displaying a scan line, averages each line with the line above and the line below. This filter is applied to all depths except direct RGB (24 bits per pixel) when the card is operating in RS-170A mode (see Figure 3).

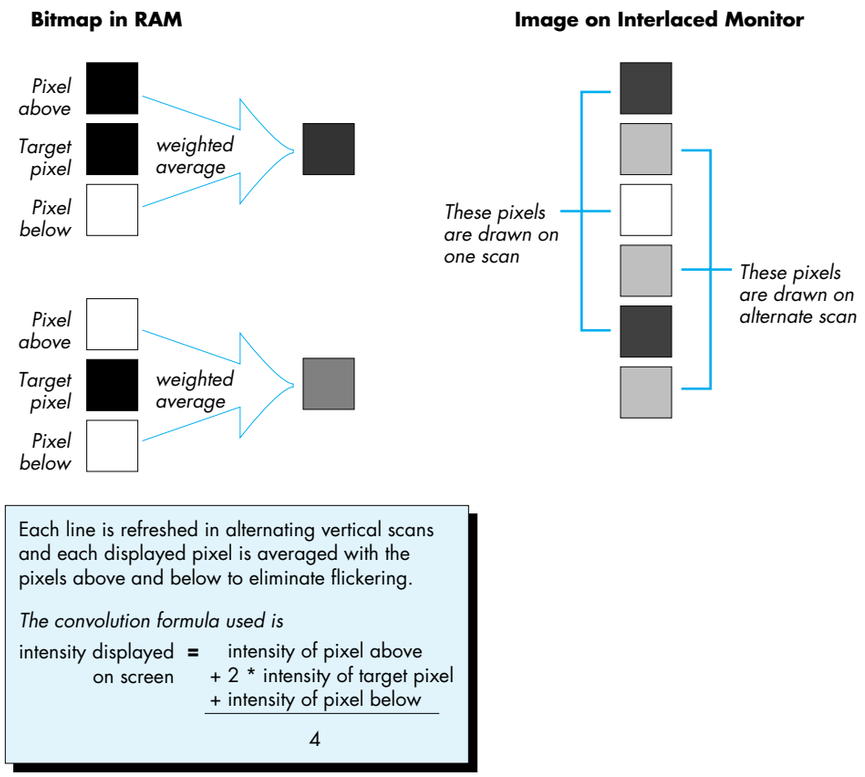


Figure 3
Convolution Filter Weights for Interlaced Video

PAL-COMPATIBLE SIGNALS

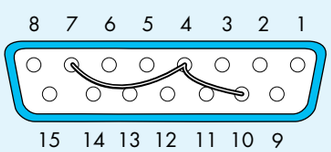
The Macintosh Display Card 8•24 GC also produces video signals that are compatible with PAL timings. These signals can be used to produce PAL output to drive video devices of the type used in most European countries.

The PAL-compatible signals have the same characteristics as the RS-170A output; that is, the output is interlaced. For settings of 1 to 8 bits deep, a convolution filter is applied before each line is produced. The PAL output mode is triggered through one of the extended sense-line configurations, as described in the accompanying sidebar, "Extended Sense Line Protocol."

EXTENDED SENSE LINE PROTOCOL

The Sense Line Protocol was implemented when Apple recognized the need for a mechanism that would allow a display card to identify the monitor connected to it. For example, the Macintosh IIci display circuitry and the Macintosh 8•24 and 8•24 GC display cards can now configure themselves according to the monitor that is connected at boot time. The identification scheme works fine, but there is one problem. Three sense lines limit the number of different monitors to seven. To overcome this limitation, newer display cards use an extension to the sense line scheme that allows for 28 new codes.

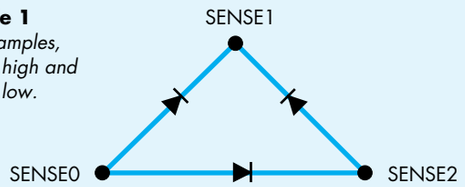
The extension is based on the following idea: When the display circuitry senses a configuration that in the original scheme signals "no display connected" (in other words, when all three sense lines are not connected), the card pulls down each sense line, one by one, and reads back what the other lines return. To return a unique code, the only requirement is that the sense lines be connected, in the cable or the monitor itself, by wires or diodes. The beauty of this idea is that existing monitors are detected correctly. Newer monitors can have their own encoding, and the circuitry for detecting new monitors is relatively simple. Since there are no active components, adding the encoding to new or existing monitors involves only a few inexpensive diodes and a little wire.



To tell the 8•24 GC card to configure itself for PAL timing output, sense lines must be connected as shown here.

Figure 4
PAL Connector

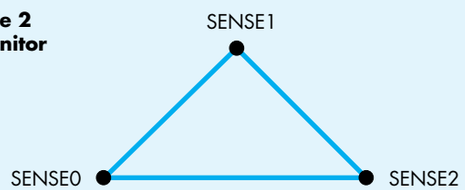
Example 1
In the examples, 1 means high and 0 means low.



The circuitry shown here produces the following code:

SENSE0 low	▶	SENSE1 1	SENSE2 1
SENSE1 low	▶	SENSE0 0	SENSE2 0
SENSE2 low	▶	SENSE0 0	SENSE1 1

Example 2
PAL Monitor



It follows that the extended sense line code for PAL is

SENSE0 low	▶	SENSE1 0	SENSE2 0
SENSE1 low	▶	SENSE0 0	SENSE2 0
SENSE2 low	▶	SENSE0 0	SENSE1 0

GC QUICKDRAW: SPIRIT STRONGER THAN FLESH

Software for the Macintosh Display Card 8•24 GC is made up of three main components (see Figure 5).

- GC QuickDraw: An optimized version of 32-Bit QuickDraw tailored for the best performance on the Am29000 processor.
- The Am29000 kernel (also referred to as GC OS): The software in the card that takes care of scheduling the execution of GC QuickDraw; also manages memory on the card.
- IPC Software: The communication mechanism between the 680x0 environment and the GC QuickDraw software. For maximum performance, the IPC software was designed to deal only with QuickDraw operations, thus limiting the overhead. The software intercepts QuickDraw drawing commands and redirects them to GC QuickDraw on the card, buffering commands when needed.

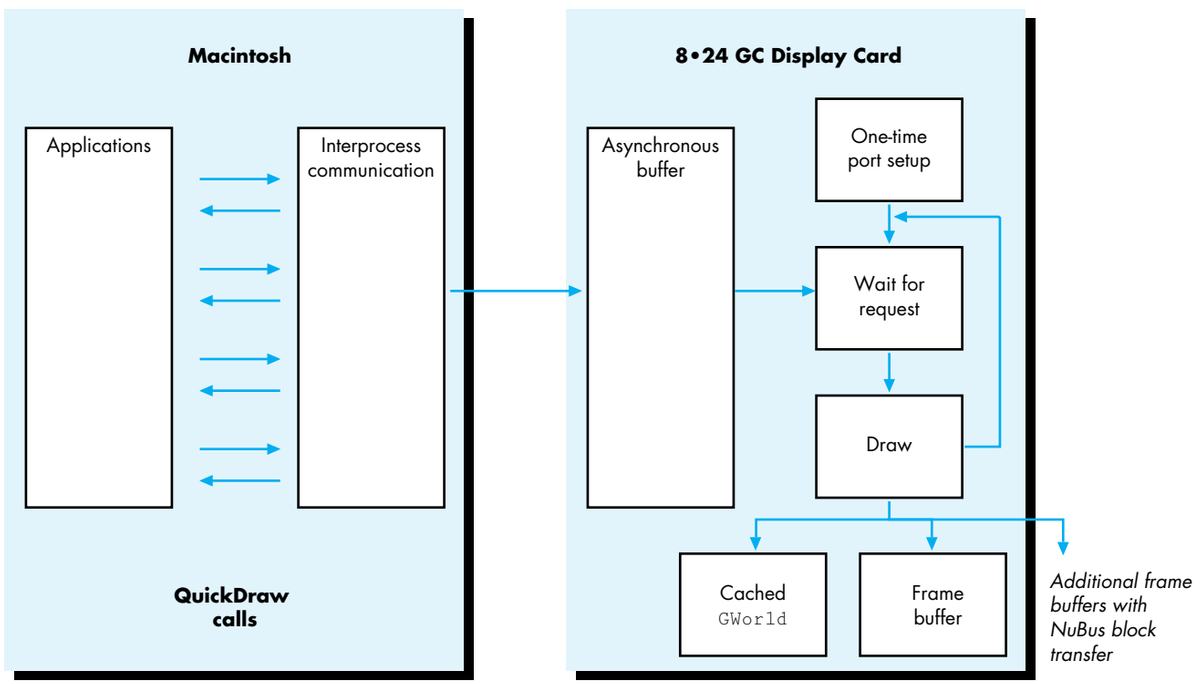


Figure 5
Software Architecture of the 8•24 GC Card

All these components come in a single file called 8•24 GC, which must be moved to the System Folder as a single-step installation process. At boot time, after the "Welcome to Macintosh" screen appears, the System looks for the file and, when it finds it, installs the GC software as a transparent upgrade to 32-Bit QuickDraw. The installation consists of loading the GC QuickDraw code into the card memory, loading the GC kernel (GC OS) and initializing its structures, and loading and initializing the IPC software. At INIT time, the rocket icon appears to indicate the presence of the GC software in the System.

The 8•24 GC file contains a Control Panel device that allows the user to turn acceleration on and off.

THE HAND IS QUICKER THAN THE EYE

A good way to start the discussion of GC QuickDraw is by separating QuickDraw calls into two global classes. The first class consists of the "drawing" commands—**`LineTo`** and **`FrameRect`**, for example. The second class consists of the "state"-oriented calls, such as **`SetPenState`** and **`RGBForeColor`**.

DRAWING CALLS

When 32-Bit QuickDraw is running the show, all drawing commands cause one of the standard QuickDraw bottleneck procedures to be called. For example, calling **`PaintRect`** causes **`StdRect`** to be called, and calling **`FillPoly`** ends up invoking **`StdPoly`**. "Under" the bottleneck procedures, QuickDraw takes the commands and goes into a set of calls that execute commands based on the type of drawing taking place. The lowest level for QuickDraw is the code that actually draws to the devices affected by the drawing that is being executed.

The software for the 8•24 GC card takes effect at the level immediately under the standard (bottleneck) procedures. If **`DrawChar`** is called, for example, QuickDraw checks to see if the **`CQDProcs`** field in the current port is **`nil`** and, if so, calls the **`StdText`** routine. At this point the IPC software intercepts the call and passes the request to GC QuickDraw to initiate the accelerated process.

The previous paragraph points to an important fact: If an application completely replaces the standard bottleneck procedures, it is effectively turning acceleration off. Ordinarily standard bottleneck procedures are completely replaced only when driving nondisplay devices such as printers, so there is no conflict with the acceleration scheme. But some applications may bypass QuickDraw completely. By doing so they forgo the benefits of accelerated QuickDraw.

When the IPC takes control, it handshakes with the card kernel, queues the command in the asynchronous buffer, and returns control to the application. From the application's point of view, the call to QuickDraw returns almost immediately

and allows the application to continue its execution. This parallelism reduces the apparent time it takes QuickDraw to perform a drawing operation. Performance is also enhanced because, while GC QuickDraw is performing the task, the application can be busy calculating whatever is necessary for the next call.

When GC QuickDraw processes one command from the queue, two paths are available. If the target is the current port, and if no changes have been made, GC QuickDraw goes ahead and does the drawing; it has in its cache everything it needs. On the other hand, if the target is a different port, or if the current port has changed, GC QuickDraw proceeds to import the necessary parameters from main memory, set its own structures, and finally do the actual drawing. In this discussion, we've used the word *port*, but the same principle applies to any change if the destination is a **GWorld** or if the **GDevice** associated with the destination has changed.

The application can issue as many drawing commands as necessary. The IPC software buffers them as needed, and GC QuickDraw completes all operations as quickly as it can. The result is a completely asynchronous operation that gives applications the fastest responsiveness and frees more processor time for application calculations (applications have more time to think!).

There are some exceptions to the complete asynchronous scheme. Calls (such as **CopyBits**) that involve not only the destination port/device combination, but also different source and destination environments, make it necessary to flush the queue by processing all pending operations and then performing the setup for the call. Once the **CopyBits** call has been initiated, it is executed in parallel for optimum performance. Software developers should note that using **GWorld**s to buffer **PixMaps** makes **CopyBits** intrinsically faster.

PARAMETER-CHANGING ("STATE") CALLS

The Macintosh Display Card 8•24 GC acceleration mechanism concerns itself mainly with the drawing class of calls. Parameter-changing calls also have an important effect on the overall performance of GC QuickDraw. The 8•24 GC software does not work in its own environment, but heavily relies on the global structures that QuickDraw keeps in main memory.

Instead of loading all the parameters with every call, GC QuickDraw caches all the data structures it needs, but has to reload the data when changes are detected. Some of the structures cached by GC QuickDraw are color tables, **GDevice**s, **GWorld**s, **PixPat**s, Fonts, and Width Tables. Changing the pen or modifying the foreground color, for example, invalidates the cached data. To squeeze the maximum out of the 8•24 GC card, it is therefore important to group calls that draw elements with shared characteristics (same color, same pattern, and so forth) instead of constantly changing port parameters between drawing calls.

Changing directly any structure maintained and used by QuickDraw has always been a clear and dangerous compatibility risk. The 8•24 GC software cannot detect changes to the drawing environment unless you use the proper calls to accomplish such changes.

Not all QuickDraw calls are redirected when acceleration is enabled. Acceleration affects only calls that either perform drawing operations or change the state of QuickDraw structures. A list of such calls follows.

SetPortBits	**PortSsize**	**MovePortTo**
SetOrigin	**SetClip**	**ClipRect**
StdLine	**StdBits**	**StdText**
StdRgn	**StdArc**	**StdRRect**
StdOval	**StdRect**	**StdPoly**
InitGDevice	**DisposGDevice**	**AddSearch**
DelSearch	**AddComp**	**DelComp**
DisposPixPat	**CopyPixPat**	**MakeRGBPat**
GWorldDispatch	**HidePen**	**ShowPen**
SetPenState	**PenSize**	**PenMode**
PenNormal	**BackColor**	**ColorBit**
OpenRgn	**OpenPicture**	**OpenPoly**
CopyRgn	**SetRectRgn**	**RectRgn**
OffsetRgn	**InsetRgn**	**SectRgn**
UnionRgn	**DiffRgn**	**XorRgn**
MapRgn	**RGBForeColor**	**RGBBackColor**
CopyPixMap	**PenPixPat**	**BackPixPat**
OpColor	**HiliteColor**	**SetPortPix**
PenPat	**BackPat**	**CopyBits**
DrawPicture	**PMgrDispatch**	

THE DEVELOPER POINT OF VIEW

The best feature of the Macintosh Display Card 8•24 GC is that application developers have little to worry about when it comes to compatibility. You must put forth some effort to do something in an application that GC QuickDraw cannot straighten out.

The one area that may require some rewriting is changing the way an application allocates offscreen drawing environments (see "Braving Offscreen Worlds" in develop, issue 1). When a **GDevice** is created by hand (instead of by a call to **NewGWorld**), all the structures are kept in main memory. Drawing to this environment and then using **CopyBits** to display the result takes a greater number of NuBus transfers.

When a drawing operation that involves a **GWorld** occurs, GC QuickDraw immediately caches the complete **GWorld** structure in the card's memory if the structure has not yet been cached and if sufficient memory is available.

(The card's optional DRAM kit is an important addition for applications that work with large **GWorld**s.) When **CopyBits** is called to display the results, the transfer of pixels to the screen driven by the 8•24 GC card is therefore really fast because there is no NuBus transfer. Even displaying the image into other monitors benefits, especially when the other cards can accept block transfers. Drawing operations to and from **GWorld**s can be executed in parallel. This is not the case when drawing to or from old-style offscreens (see Figures 6 and 7).

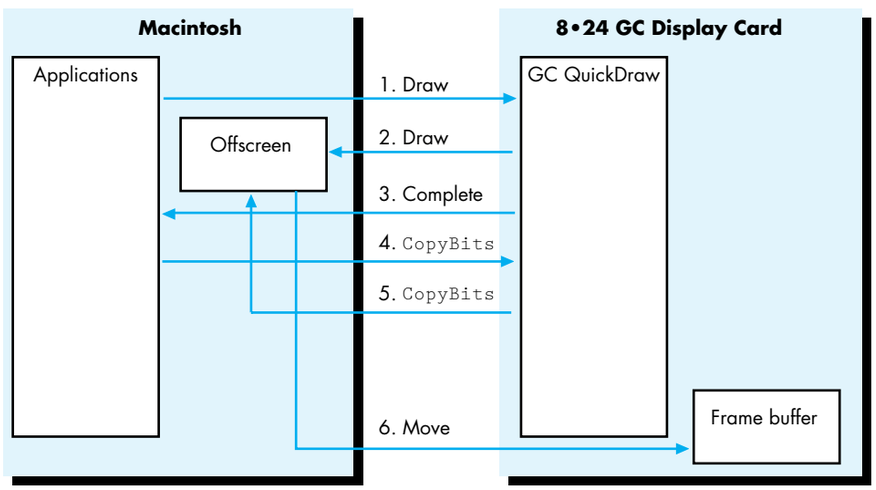


Figure 6
Offscreen Old Style

When applications create offscreen environments by hand, GC QuickDraw performance is affected because the data has to go many more times across NuBus. For example, if an application draws to an offscreen port and then calls **CopyBits** to display the results on the screen, the following happens:

1. IPC passes the drawing command to GC QuickDraw.
2. GC QuickDraw does the drawing across the Nubus to buffer.
3. The application waits for the completion signal.
4. IPC passes the **CopyBits** call to GC QuickDraw.
5. GC QuickDraw copies from main memory…
6. …to the frame buffer.

Finally, QuickDraw operates more efficiently when the **PixMap**'s **rowBytes** is a multiple of four (long word alignment). It just happens that when **NewGWorld** is called it creates the offscreen **PixMap** in such a fashion, which increases the performance of QuickDraw even when the offscreen buffer could be in main memory.

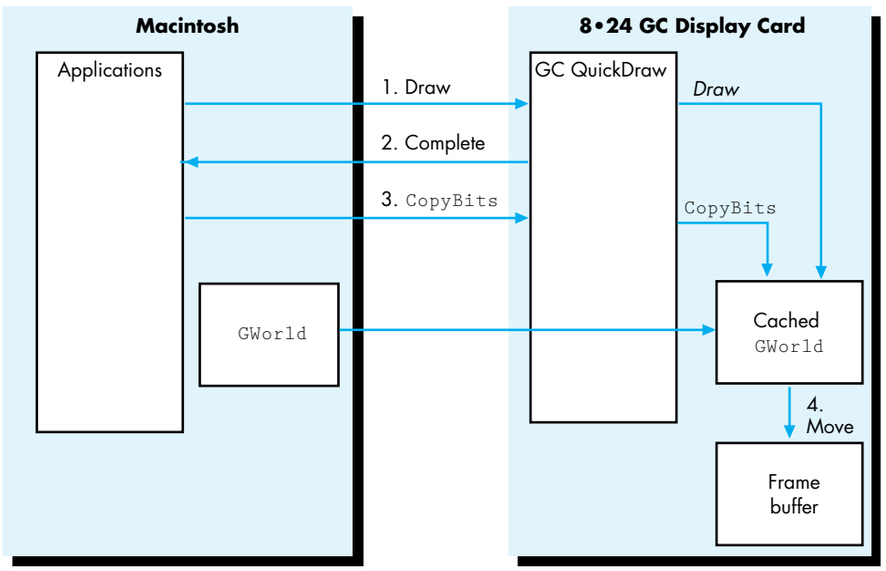


Figure 7
Offscreen **GWorld**s

When an application uses **GWorld**s to buffer the display, GC QuickDraw can use the cached data in its memory and minimize NuBus traffic. The steps are as follows:

1. IPC passes the draw command and returns control to the application immediately.
2. GC QuickDraw executes the command as soon as possible.
3. IPC passes the **CopyBits** call and returns.
4. GC QuickDraw moves pixels from the local copy of the **GWorld** to the screen buffer while the application does its stuff.

COMPATIBILITY ISSUES

As mentioned earlier, GC QuickDraw is port oriented, which suggests that to improve performance it caches the port's structure in its entirety while drawing to a port. The implication is that, if an application draws to different ports in an alternating fashion, the application is forcing GC QuickDraw to flush its asynchronous buffer and move data from main memory with each drawing command.

To get the best performance from GC QuickDraw, a good programming technique is to bundle all drawing operations that affect a given port and complete them before changing ports to do some more drawing. As noted earlier, it is also important to put together all the calls that affect a single port and share characteristics such as color, pattern, and pen. The same admonition applies to **GWorld**s because

GC QuickDraw caches the **GWorld** into its memory when a **GWorld** is the destination of a drawing operation. Changing **GWorld**s and drawing to each one in rapid succession means that GC QuickDraw has to keep flushing its asynchronous buffer and moving the necessary data from memory, which could imply copying the new **GWorld**, including its **PixMap** (and 32-bit **PixMap**s can be a lot of bytes to move).

BOTTLENECKS AND STRUCTURE CHANGES

As mentioned earlier, two areas make life difficult for an application running under GC QuickDraw. These areas are replacing the bottleneck procedures and changing QuickDraw structures directly.

Replacing bottleneck procedures. Replacing the standard bottleneck procedures is a time-honored tradition in the Macintosh world. The problem is that, if an application replaces a standard procedure and does not end up calling the original default trap after it finishes its manipulations, the application is turning off acceleration. Whenever possible—and if acceleration is desired—the application that replaces the standard bottleneck procedures should call the original through the trap dispatcher before returning to the main program.

Changing QuickDraw structures. The dangers of messing up with QuickDraw structures directly are well known to all. Nevertheless, some applications still go about happily changing structures. As a rule, you should call the ToolBox whenever possible to change the state of the drawing environment. When the need is too strong, use the calls that alert QuickDraw to the changes you're making.

TIMING

One interesting compatibility problem is not uniquely tied to the 8•24 GC card but has to do with faster hardware in general. Some techniques still being used are direct descendants of routines that were implemented for the 64K Macintosh. Now, with new machines, these techniques simply run too fast.

One example is the "marching ants" technique of repeatedly calling **FrameRect**, with each repetition shifting a pattern 1 bit to create a rippling frame on the screen. The trick is still valid, but now causes a problem. In the time it took the Ol' Macintosh to make four passes through the **FrameRect** loop, a newer Macintosh can make hundreds of passes. With 8•24 GC acceleration on, the number is even greater. The result is that the marching becomes jerky and annoying, and the cursor tends to disappear, because **HideCursor** is being called to complete the **FrameRect** call. The cursor appears and disappears so quickly that it becomes invisible.

Do not time loops based on the performance of a given machine. For any form of animation, use the Time Manager routines to time your application.

ALERTING QUICKDRAW TO CHANGES

Applications can use the calls listed here to make QuickDraw and GC QuickDraw aware of direct changes to forbidden structures.

`PROCEDURE CTabChanged (ctab: CTabHandle)`
This call says, "Yes, I know I shouldn't mess with a color table directly, but I did it, and I want to come clean." Use **`SetEntries`**, or—even better—let the Palette Manager maintain the color table for you.

`PROCEDURE PixPatChanged (ppat: PixPatHandle);`
Use this call to say, "I admit I modified the fields of a **`PixPat`** directly. Please fix the resulting mess for me." When the modifications include changing the contents of the color table pointed to by **`PixPat.patMap^^.pmTable`**, you should also call **`CTabChanged`**.

`PenPixPat` and **`BackPixPat`** are better ways to install new patterns.

`PROCEDURE PortChanged (port: GrafPtr);`
You should not modify any of the port structures directly. But if you cannot control yourself, use this call to let QuickDraw know what you have done.

If you modify either the **`PixPat`** or the color table associated with the port, you need to call **`PixPatChanged`** and **`CTabChanged.`**

`PROCEDURE GDeviceChanged (gdh: GDHandle);`
The best practice is to stay away from the fields of any **`GDevice`**. But if you do change something, make this call to rectify any problems. If you change the color table data in the device's **`PixMap`**, you must also call **`CTabChanged.`**

DRAWING DIRECTLY TO THE SCREEN

As a final admonition, *do not draw directly to the screen*. Drawing directly to the screen puts any application at immediate compatibility risk with 32-Bit QuickDraw and with all Apple and third-party 24-bit video cards. Drawing directly into the 8•24 GC card's display buffer will probably cause crashes and, at the minimum, create weird artifacts if both the application and GC QuickDraw are drawing at the same time.

PROGRAMMING THE Am29000 (NOPE!)

No, you can not program the Am29000 yourself. At the moment of this writing, third-party applications cannot make direct use of the Am29000 processor for any purpose.

The good news is that Apple recognizes the burning desires developers have to use the Am29000 for more than accelerating QuickDraw, and future releases of the software may provide such capability. So if you have a brilliant idea, Apple wants to know about it. Although implementation details are not yet available, you can start by sending a message that describes what you want to do and what kind of support you hope the GC software can give you.

Send these and any other comments about programming the Am29000 on the 8•24 GC card to AppleLink address FAST.GRAPHIC. Apple will get back to you when more information is available.

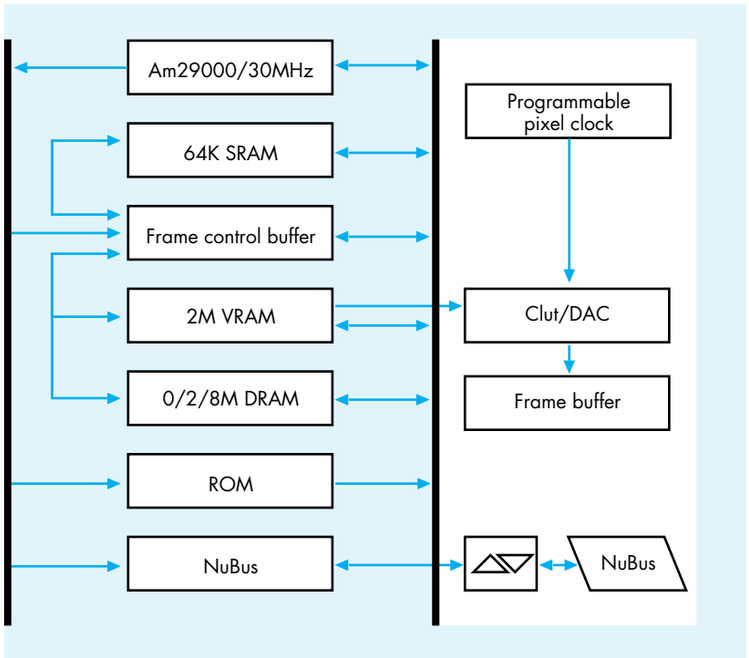


Figure 8
Block Diagram of the 8•24 GC Card

NOTEWORTHY CALLS

One piece of software that has not been mentioned is the 8•24 GC **`cdev`**. With it users can turn acceleration on and off through the Control Panel. Although turning acceleration on and off should be left under the control of the user, you may want to provide users with some kind of on/off dialog box or menu option. The *Developer Essentials* disc (this issue) contains the **`GraphAccel.o`** file, an MPW library that accomplishes this task. A sample FKey for turning acceleration on and off is included with the library.

For More Information

Designing Cards and Drivers for the Macintosh Family, second edition, Addison-Wesley, 1990, available from APDA.

Macintosh Display Card 8•24 GC, Developer Notes, April 1990, available from Developer Technical Publications.

A new call was implemented with 32-Bit QuickDraw to allow an application to determine whether a drawing operation initiated by it, and affecting a given port, has been finished:

```
Function QDDone(pPtr: CGrafPtr): Boolean;
```

This call may be of especially good use if you're interested in timing each individual call. It may also be of use in some animation situations, such as when you want to initiate the next drawing action only after execution of a prior call has definitely concluded.

QDDone returns FALSE if QuickDraw is in the process of drawing to **pPtr** and TRUE when QuickDraw is done. If **pPtr** is **nil**, **QDDone** returns TRUE only when drawing to all ports is completed. Note that when **nil** is passed to the call, background processes such as clocks could prevent **QDDone** from ever returning TRUE.

```
Function QDDone(pPtr: CGrafPtr): Boolean;
INLINE $203C, $0004, $0013, $AB1D;  { Move.l #$00040013,D0
                                      _QDOffscreen
                                    }
```

FINALLY

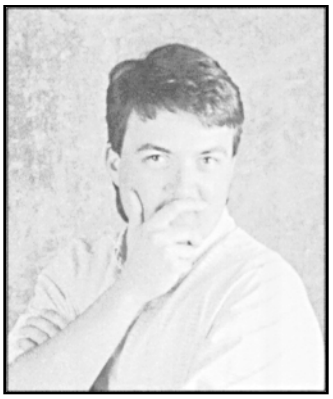
The Macintosh Display Card 8•24 GC is Apple Engineering's response to the demands for enhanced graphics performance for the Macintosh family of computers. Designers of graphics-intensive applications now can concentrate on the algorithms that will improve overall performance instead of using design resources to squeeze the last drops of display speed, a choice which often compromises compatibility and ease of use.

Developers get the benefits of the 8•24 GC card with almost no programming hit. Coding of special routines to achieve faster performance is unnecessary. Following old, simple QuickDraw rules, developers can be assured of faster graphics performance at no extra cost.

Thanks to Our Technical Reviewers:
Dave Cho, Casey King, Mark Krueger,
Jean-Charles Mourey

MEET PRGENERAL, THE TRAP THAT MAKES THE MOST OF THE PRINTING MANAGER

The Printing Manager has been expanded and enhanced by the addition of the trap PrGeneral. This little-known trap—available in ImageWriter® driver versions 2.5 and later, and LaserWriter® driver versions 4.0 and later—can pass five operation codes that solve special problems and improve a printer's performance. This article describes the trap and its opcodes. The accompanying sample application on the Developer Essentials *disc enables you to experiment with the opcodes as you print images we provide.*



PETE "LUKE" ALEXANDER

The little-known trap `PrGeneral`, through its five operation codes, can give your application the ability to achieve highest-resolution print output, verify page orientation, and increase performance by avoiding the need to spool. These enhancements to the Printing Manager come in handy in a variety of situations.

The sample application PrGeneral Play lets you print images with and without the `PrGeneral` opcodes. That way you can compare the images and see for yourself the effects of the opcodes. We'll look at fragments of the sample application in the course of this article.

ABOUT PRGENERAL

`PrGeneral` is a multipurpose call that can perform a number of different functions, depending on the opcode used with it. `PrGeneral` currently can pass five opcodes: `GetRslData`, `SetRsl`, `GetRotn`, `DraftBits`, and `NoDraftBits`.

- GetRslData enables an application to determine the resolutions that the currently selected printer supports.
- SetRsl specifies the resolution to print with, so that your application can achieve the highest-resolution print output.

PETE "LUKE" ALEXANDER loves technical support for printing because it's "hard, ugly, and kinda sick." He earned his middle name while using The Force to feel his way through the stickier parts of the art. Naturally, he feels quite at home with light sabers. He's been at Apple for two years, after doing a brief stint at a company he won't name (hint: BLUE). He's a born-and-bred Silicon Valley boy. He rides mountain bikes, sails a 16-foot Hobie Cat, and chuckles over Calvin and Hobbes. He loves to get high—30,000 feet to be exact, in an airplane without an engine (read: glider). He established a few records for that possibly unequaled but arguably looney feat. If he ever asks you if you need a ride, please think twice, and if you say yes, may The Force be with you. •

- `GetRotn` enables an application to determine if the landscape orientation has been selected in the style dialog, useful if your image will only fit on a page when printed in landscape orientation.
- `DraftBits` forces draft printing, thereby avoiding the need to spool large quantities of data to disk, while also enabling printing of bitmaps and pixel maps.
- `NoDraftBits` cancels the effect of `DraftBits`.

`PrGeneral` is declared like this in C:

```
pascal void PrGeneral (Ptr pData);
```

The `pData` parameter is a pointer to a record called `TGnlData`. The first eight bytes comprise a header shared by all the `PrGeneral` calls:

```
struct TGnlData {
      short iOpCode;
      short iError;
      long        lReserved;
};
```

The first field in the record, `iOpCode`, contains the opcode that is passed through the call to `PrGeneral` to obtain the requested feature. The second field, `iError`, contains the result code returned by the call to `PrGeneral`. The final field, `lReserved`, is reserved for future use by the Printing Manager and/or the Printer Driver. Additional fields follow `lReserved`, depending on the opcode that is used.

After each call to `PrGeneral`, your application should check the value in the `iError` field. Three possible result codes can be returned:

```
#define noErr       0   /* You've seen this one before. */
#define NoSuchRsl   1   /* Only defined for PrGeneral. */
#define OpNotImpl   2   /* Only defined for PrGeneral. */
```

If `PrGeneral` accomplishes your request, it returns `noErr` in the `iError` field. If you request a resolution that is not supported by the currently selected printer, the call to `PrGeneral` returns the `NoSuchRsl` error code. Finally, some printer drivers might not support one of the opcodes described here, in which case the call to `PrGeneral` returns the `OpNotImpl` error code. ImageWriter driver versions 2.5 and later, and LaserWriter driver versions 4.0 and later support all of the `PrGeneral` opcodes.

Your application should also check `PrError` (which returns the result code left by the last Printing Manager routine) after checking `iError`, to be sure that no additional error was generated by the Printing Manager or the Printer Driver. See Technical Note #72, Optimizing for the LaserWriter, for a complete list of the possible result codes returned by the Printing Manager or the LaserWriter driver.

If `resNotFound` is returned by `PrError`, then the current Printer Driver doesn't support `PrGeneral`. This shouldn't be a major problem for your application, but your application must be prepared to deal with this error. If you do receive the `resNotFound` error back from `PrError`, you should clear the error with `PrSetError(0)`; otherwise, `PrError` might still contain this error the next time you check it.

If an error is returned by `PrError`, be sure that all of the Printing Manager calls receive their corresponding close calls before you report the error to the user. This enables the Printing Manager and the Printer Driver to clean up their worlds before you exit. See Technical Note #161, When to Call PrOpen and PrClose, for a demonstration of the technique.

If `noErr` is returned by `PrError`, you can then proceed.

ACHIEVING HIGHEST-RESOLUTION OUTPUT

Your application can use the `SetRsl` opcode to set print resolution to the highest supported resolution of the current printer. But before doing this, it needs to determine the resolutions supported by the current printer, using the `GetRslData` opcode. The data returned by `GetRslData` is essential because there are now over 50 different models of printers that can be connected to the Mac, each with its own unique imaging capabilities. `GetRslData` saves your application from having to make assumptions about which resolution would or would not work.

To illustrate the benefits of setting print resolution to the highest supported resolution of the current device, let's compare the two graphs shown in Figures 1 and 2, printed on a LaserWriter II/NTX.

As you can see, the results without using `SetRsl` are not too impressive. (The graph is printed at 72 dpi.) The same graph printed using `SetRsl` looks quite a bit better. (The graph is printed at 300 dpi.) You can demonstrate the effects of using `SetRsl` for yourself with the sample application PrGeneralPlay on the *Developer Essentials* disc.

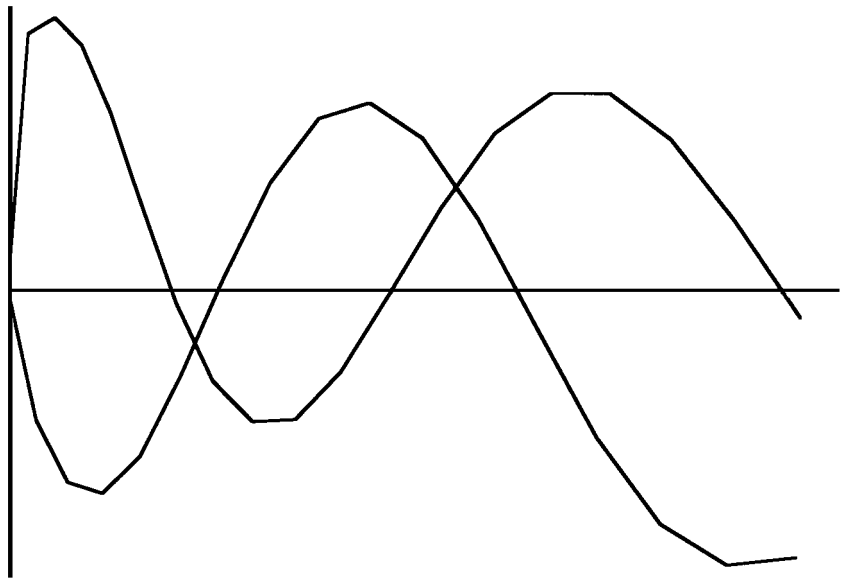


Figure 1
A Graph Printed Without Using SetRsl

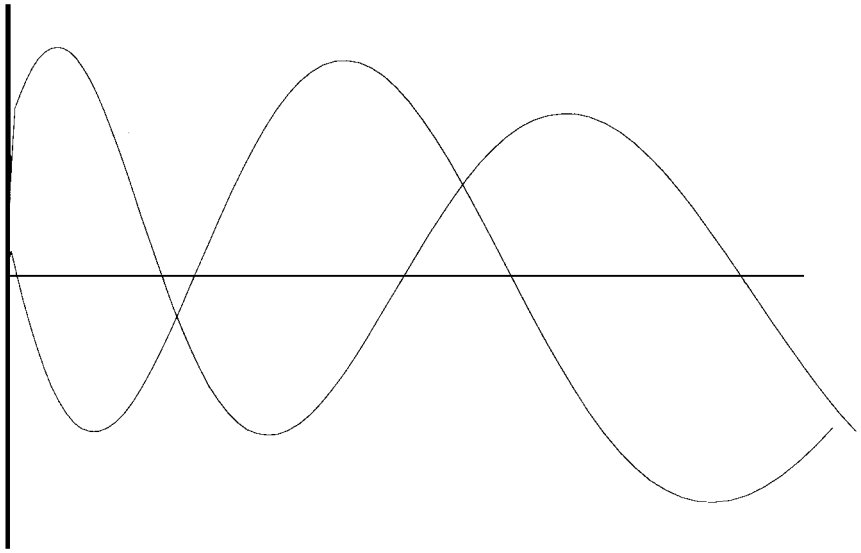


Figure 2
The Same Graph Printed Using SetRsl

USING GETRSLDATA

`GetRslData (iOpCode = 4)` requests that the Printer Driver return resolution information about the current printer. Three records are used to convey the resolution information: `TRslRg`, `TRslRec`, and `TGetRslBlk`. We'll look at these records in detail after some basic information about resolution.

A printer supports either discrete or variable resolution. Discrete resolution means that the application can choose from a limited number of resolutions (expressed in dots per inch, or dpi, in the X and Y directions) predefined by the Printer Driver. For example, the ImageWriter driver supports four discrete resolutions: 72 x 72 dpi, 144 x 144 dpi, 80 x 72 dpi, and 160 x 144 dpi. If a printer supports variable resolution, the application can define any resolution within a range bounded by minimum and maximum values. The LaserWriter driver supports variable resolution within a range from 25 dpi to 1500 dpi in both the X and Y directions.

Best quality output is always obtained by choosing a square resolution, meaning one in which the resolutions for the X and Y directions are equal. Some devices support nonsquare resolutions–that is, where the resolution for the X direction does not equal the resolution for the Y direction–but using a nonsquare resolution will result in distortion of the printed image.

Let's look now at the records that convey resolution information.

```
struct TRslRg {
      short iMin;
      short iMax;
};
```

The `TRslRg` record returns information about the resolution supported by the current printer. If the printer supports only discrete resolutions, which is the case for the ImageWriter, `iMin` and `iMax` are set to 0. Otherwise, if the printer supports variable resolution, as the LaserWriter does, these fields are set to the minimum and maximum resolutions supported.

```
struct TRslRec {
      short iXRsl;
      short iYRsl;
};
```

The TRslRec record specifies a discrete resolution supported by the printer. The iXRsl field specifies the discrete resolution for the X direction, and iYRsl for the Y direction. A printer driver can have up to 27 separate TRslRec resolution records. The ImageWriter driver contains 4 such records, which are returned when you use the GetRslData opcode. In this case, your application will need to choose one of these records to be used by SetRsl. Our application PrGeneral Play when printing to an ImageWriter uses the highest square resolution that the ImageWriter supports, which is 144 x 144 dpi.

```
struct TGetRslBlk {
      short       iOpCode;
      short       iError;
      long        lReserved;
      short       iRgType;
      TRslRgxRslRg;
      TRslRgyRslRg;
      short       iRslRecCnt;
      TRslRec     rgRslRec[27];
};
```

The TGetRslBlk record is the complete structure passed to PrGeneral when using the GetRsl opcode. It contains the iOpCode, iError, and lReserved fields, which we've already discussed, plus some others.

iRgType is a version number returned by the Printer Driver. The version number is all your application needs to determine that a particular set of functionality is or is not present. The LaserWriter and the ImageWriter will always return 1. If it's not 1, don't use the data.

xRslRg and yRslRg are the resolution ranges supported for the X and Y directions by a variable-resolution printer. If the current printer doesn't support variable resolution, the value in these fields is 0.

iRslRecCnt returns the number of resolution records used by a particular printer driver. As mentioned earlier, up to 27 are allowed.

rgRslRec is an array of resolution records, each specifying a discrete resolution at which the current printer can print an image. In the arrays returned by Apple printer drivers, the last record represents the highest supported resolution. We recommend that other printer drivers do the same.

The records shown in Figure 3 are returned by `PrGeneral` for the LaserWriter and the ImageWriter, respectively.

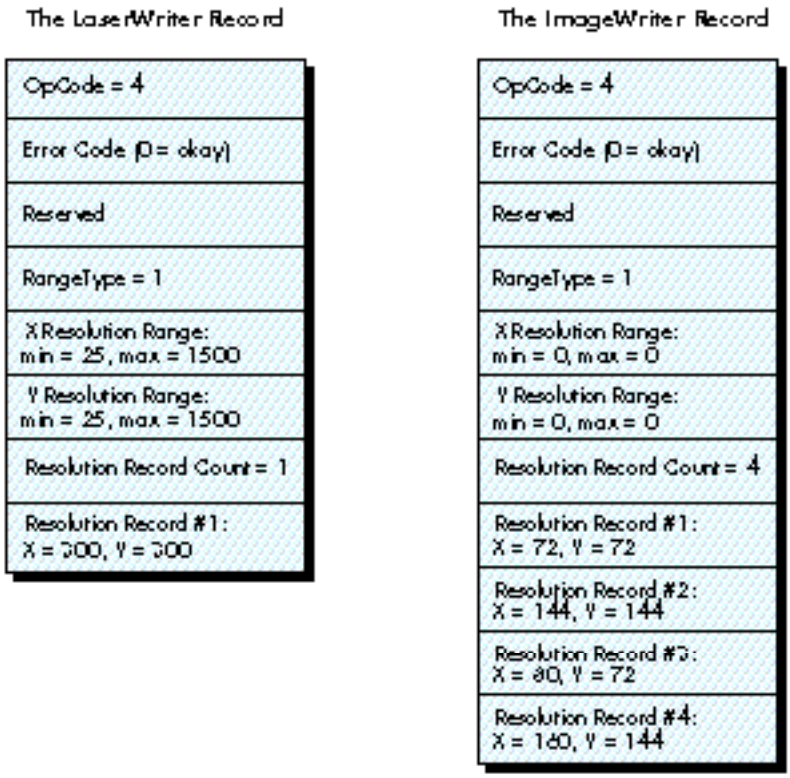


Figure 3
The Records for the LaserWriter and the ImageWriter

Note that in the LaserWriter record, the resolution range shown is 25 through 1500 dpi. *Inside Macintosh*, volume V, page 413, shows the minimum value as 72; this is an error. And although up to 1500 dpi is supported by the Printer Driver, the device itself is only capable of a maximum resolution of 300 x 300 dpi. Accordingly, the single resolution record indicates that the printer will only support a maximum resolution of 300 x 300 dpi. Other devices can achieve higher resolutions, up to the maximum supported by the driver.

In the ImageWriter record, all the resolution range values are 0, because the printer only supports discrete resolutions. The four resolution records returned give your application the option to choose one of these discrete resolutions. Note that the highest supported resolution is represented by the last record.

USING SETRSL

`SetRsl (iOpcode = 5)` tells the Printer Driver the desired imaging resolution requested by the application. The contents of the record are as follows:

```
struct TSetRslBlk {
      short       iOpCode;
      short       iError;
      long        lReserved;
      THPrint     hPrint;
      short       iXRsl;
      short       iYRsl;
};
```

We have already discussed the `iOpCode`, `iError`, and `lReserved` fields, so we'll start with the `hPrint` field. `hPrint` contains a handle to a print record that has previously been created and passed through `PrDefault` to make sure that all of the information contained in the handle is good. If you are using a print record that was saved as a resource, you will want to call `PrValidate` on it to make sure that the contents of the handle will work with the current version of the Printing Manager and the printer driver. Because the `SetRsl` opcode may require the Printer Driver to change the appearance of the style and/or job dialogs, we want to determine and set the resolution before the print dialogs are presented to the user. This is why we need a good handle—the same handle that is passed to the dialogs.

The `iXRsl` and `iYRsl` fields contain the resolutions that you would like the Printer Driver to image with. If `iError` returns a value of 0 (`noErr`), the print record will be updated with this new resolution, which can be used at print time. If the requested resolution isn't supported by the current printer, `iError` will return `NoSuchRsl`, and the printer driver will revert to the previous setting.

You can undo a previous call to `PrGeneral` with the `SetRsl` opcode, by calling `PrGeneral` with the `SetRsl` opcode again, this time with the original resolutions used by the Printer Driver before your call to `SetRsl`. (*Inside Macintosh*, volume V, page 414, suggests making another call that specifies an unsupported resolution, such as 0 x 0. This doesn't work.) If you save the resolutions contained in the `iVRes` and `iHRes` fields of the `TPrinfo` record, you can then pass these values in the `iXRsl` and `iYRsl` fields of the `TSetRslBlk` record, and the next time you call `PrGeneral` with the `SetRsl` opcode, your resolution for the current printer will be back to its default setting. You can also call `PrintDefault` on the print record passed to the call to `PrGeneral` with the `SetRsl` opcode. This definitely works, but it loses all of the user's selections from her or his last trip to the style dialog, which is not very user-friendly!

Note that if the resolution is set to greater than 600 x 600, then the LaserWriter driver limits the reduction factor to 60 percent—that is, it will not allow you to go below that. This is done so that the page rect/paper rect coordinates fit within 16-bit signed integers.

GetRslData and SetRsl in action

The following code fragment from our sample application PrGeneral Play uses the `GetRslData` opcode of `PrGeneral` to find the highest square resolution supported by the current printer. It then sets the resolution with `SetRsl`. Note that for simplicity this code fragment and others in this article assume that `PrGeneral` and the particular opcode are supported. PrGeneral Play checks for `PrGeneral` and the opcode before using them.

```
int SetMaxResolution (thePrRecHdl)
THPrint     thePrRecHdl;
{
      int         maxDPI = 0,
                  resIndex;

      TGetRslBlk  getResRec;
      TSetRslBlk  setResRec;

      getResRec.iOpCode = getRslDataOp;
      PrGeneral ((Ptr)(&getResRec));

      /* At this point, we have an array of possible resolutions. After checking
      for errors, we loop through each resolution range record looking for the
      highest resolution available,where x and y are equal. This loop makes no
      assumptions about the order of the resolution records. */

      if (getResRec.iError == noErr && PrError() == noErr)
        {
         for (resIndex = 0; resIndex < getResRec.iRslRecCnt; resIndex++)
             {
             if ( getResRec.rgRslRec[resIndex].iXRsl ==
                  getResRec.rgRslRec[resIndex].iYRsl &&
                  getResRec.rgRslRec[resIndex].iXRsl > max DPI)

              maxDPI = getResRec.rgRslRec[resIndex].iYRsl;
             }

            /* We now have the desired resolution. If it is not zero, we use
            SetRsl to set it. */
```

```
            if (maxDPI != 0)
             {
               setResRec.iOpCode = setRslOp;
               setResRec.hPrint = thePrRecHdl;

               setResRec.iXRsl = maxDPI;
               setResRec.iYRsl = maxDPI;
               PrGeneral ((Ptr)(&setResRec));
             }

            if (setResRec.iError == noErr && PrError() == noErr
                && maxDPI != 0)
              return (maxDPI);
      }
      else return (0);
}
```

Now that the resolution has been set to the highest supported resolution of the current device, what about drawing objects? In the case of the LaserWriter, because the printer supports a physical resolution that is roughly four times higher than the screen's, your printing grafPort is now four times bigger than a standard printing grafPort. If you do not compensate for this change, your objects will be printed at micro size, as illustrated in Figure 4.

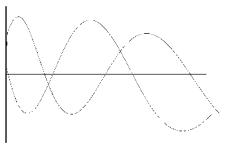


Figure 4
A Graph Printed Without Correct Scaling

You should scale all of your objects bigger by the scale factor arrived at by dividing your device resolution by your screen resolution. In the case of the LaserWriter, all of your objects, plus coordinates and font sizes, should be scaled roughly four times bigger than their original size.

The following code fragment from our sample program demonstrates this scaling idea:

```
deviceRes = SetMaxResolution (thePrRec);
```

Upon our return from using `SetMaxResolution`, we receive the highest resolution supported by the current printer. We can now determine the scaling factor:

```
if (deviceRes != 0) scale = deviceRes / MacScreenRes;
```

If the device resolution is 0, we assume that all drawing will occur at the Macintosh computer's screen resolution of 72 dpi. The value of scale was originally set at 1 when it was declared, thereby preventing problems that might otherwise occur if the `SetRsl` call fails or the driver does not support the `GetRslData` or `SetRsl` opcodes.

We then draw all of our objects bigger as specified by the scaling factor. For example:

```
TextSize (FontSize * scale);
```

VERIFYING PAGE ORIENTATION

At times it can be very useful for your application to be able to determine which page orientation the user has selected in the style dialog. For instance, if the user has not selected landscape orientation, and an image will only fit on a page when printed in landscape orientation, your application can remind the user to select this orientation to print the image. Otherwise, the user will get a clipped image.

`GetRotn` (`iOpcode = 8`) enables your application to determine if the user has selected the landscape orientation in the style dialog. This opcode should be used after the style dialog has been presented. At this point, if landscape orientation has not been selected but would give the best printed results for the current image, you can present a dialog that asks if the user wants to select landscape orientation. If the answer is yes, you should close up the print loop and start the process over again. If the answer is no, you can then proceed with printing the image.

`GetRotn` has been implemented in the ImageWriter and LaserWriter drivers. Here are the contents of the record:

```
struct TGetRotnBlk {
       short      iOpCode;
       short      iError;
       long       lReserved;
       THPrint    hPrint;
       Boolean    fLandscape;
       char       bXtra;
};
```

If the landscape orientation has been selected by the user in the style dialog, then `fLandscape` is true. The final field in this record, `bXtra`, is reserved for future use by the Printing Manager and/or the Printer Driver.

The following code fragment from our sample program uses the `GetRotn` opcode to determine if the user has selected landscape orientation from the style dialog:

```
Boolean IsLandscapeModeSet (thePrRecHdl)

      THPrint           thePrRecHdl;
{

      TGetRotnBlk       GetRotRec;

      GetRotRec.iOpCode = getRotnOp;
      GetRotRec.hPrint = thePrRecHdl;

      PrGeneral ((Ptr) &GetRotRec);
      /* We now have the result from our call to PrGeneral, but we check
      all known errors to make sure that PrGeneral was successful and
      no errors have been encountered from printing land. */

      if (GetRotRec.iError == noErr && PrError() == noErr
        &&  GetRotRec.fLandscape)
        return (true);
      else return (false);
}
```

FORCING IMMEDIATE PRINTING TO AVOID SPOOLING

If your application needs to print only text or bitmaps, it can increase performance and save disk space by printing in draft mode. `DraftBits(iOpcode = 6)` forces draft printing. This means that the document will be printed immediately, rather than spooled to disk, as in spool printing. In the latter mode, the Printing Manager writes out a representation of the document's printed image to a disk file; this information is then converted into a bit image and printed. On the ImageWriter, draft printing is used to print quick, low-quality drafts; spool printing is used for standard or high-quality printing. The LaserWriter always prints in draft mode.

With `DraftBits`, you can print bitmaps via calls to `CopyBits`. (Normally, in draft mode, bitmaps and pixel maps are ignored.) Note, though, that the landscape orientation is not available when printing with `DraftBits`; and this opcode does not have any effect if the printer only prints in draft mode (like the LaserWriter), does not support draft printing, or does not print bitmaps.

Here are the contents of the record:

```
struct TDftBitsBlk {
     short       iOpCode;
     short       iError;
     long        lReserved;
     THPrint     hPrint;
};
```

We've already discussed the `iOpCode`, `iError`, and `lReserved` fields. The `hPrint` field is discussed in the section on using `SetRsl`.

Using this opcode may require the Printer Driver to change the appearance of the style and job dialogs. In the case of the ImageWriter, using the `DraftBits` opcode before presenting the dialogs to the user disables the landscape icon in the style dialog, and the Best and Faster options in the job dialog. The `DraftBits` opcode can also be used after the call to the job dialog, to give users the choice of print quality without forcing draft printing on them, but if the user chooses draft printing from the job dialog, this will prevent the printing of any bitmaps or pixel maps in the document. Therefore, you may prefer to use `DraftBits` before presenting the dialogs.

You should keep one additional point in mind when using the `DraftBits` opcode: all of the data that is printed must be Y-sorted, because reverse paper motion is not possible on the printer when printing your image in draft mode. This means that you cannot print two objects side by side. That is, the top boundary of an object cannot be higher than the bottom boundary of the previous object. If you violate this requirement, you will get some extremely undesirable results. To get around this restriction, you should sort your objects before print time.

This code fragment from our sample program demonstrates the use of the `DraftBits` opcode to force immediate draft printing:

```
THPrint doDraftBits (thePrRecHdl)
THPrint     thePrRecHdl;
{
     TDftBitsBlk       draftBitsBlk;
     draftBitsBlk.iOpCode = draftBitsOp;
     draftBitsBlk.hPrint = thePrRecHdl;

     PrGeneral(&draftBitsBlk);

     if ((PrError() == noErr) &&
            (draftBitsBlk.iError == noErr))
            return (true)
     else return (false)
}
```

At this point, the code returns the result to the calling function. If DraftBits was set without any problems, we return true. Otherwise, an error occurred and we return false.

You use the NoDraftBits opcode to turn off the DraftBits opcode. The contents of the record are the same as for DraftBits. If you call NoDraftBits without first calling DraftBits, this opcode does nothing.

Here is a code fragment from our program that uses the NoDraftBits opcode to turn off draft printing:

```
THPrint doNODraftBits (thePrRecHdl)
THPrint     thePrRecHdl;

{
      TDftBitsBlk       draftBitsBlk;

      draftBitsBlk.iOpCode = nodraftBitsOp;
      draftBitsBlk.hPrint = thePrRecHdl;

      PrGeneral(&draftBitsBlk);
      if ((PrError() == noErr) &&
            (draftBitsBlk.iError == noErr))
            return (true)    /* DraftBits is on. */
      else return (false)     /* DraftBits is NOT on. */
}
```

At this point, the code returns the result to the calling function. If NoDraftBits was set without any problems, we return true. Otherwise, an error occurred and we return false.

THINGS TO REMEMBER WHEN USING PRGENERAL

We have looked at the five opcodes currently available in the PrGeneral trap. GetRslData and SetRsl are used to determine and set the resolution of the printer, thus enabling your application to achieve highest-resolution output. GetRotn enables your application to determine if the landscape orientation has been selected in the style dialog. DraftBits is used to force draft printing, thus avoiding the need for spooling, while also enabling printing of bitmaps and pixel maps; and NoDraftBits cancels the effect of DraftBits.

In review, here are the things you should always keep in mind when using the PrGeneral trap:

- `PrGeneral` has been implemented in ImageWriter driver versions 2.5 and later, and LaserWriter driver versions 4.0 and later. You should check for the Resource Manager error `resNotFound` after the first call to `PrGeneral` to see if `PrGeneral` is implemented in the Printer Driver in use. If you receive this error, `PrGeneral` is not implemented.
- Your application should always check `iError` in the `TGnlData` record after making the call to `PrGeneral`, thereby ensuring that the call completed correctly. Your application should also check `PrError` before proceeding. Technical Note #72, Optimizing for the LaserWriter, contains a complete list of result codes returned by the Printing Manager and the LaserWriter driver.
- `GetRsl`, `SetRsl`, and `NoDraftBits` should always be called before the style and job dialog boxes are presented to the user; `DraftBits` should preferably be called before the dialogs are presented, although it can also be called after; and `GetRotn` should always be called after the dialogs are presented.
- The `DraftBits` opcode will have no effect if the printer always prints in draft mode, does not support draft printing, or does not print bitmaps.

To reinforce what you've learned here, you can experiment with `PrGeneral` by printing images with the PrGeneral Play program on the *Developer Essentials* disc.

Thanks to Our Technical Reviewers:
Jay Patel, Larry Rosenstein,
Scott "ZZ" Zimmerman

INDEX

For a complete source code listing and a cumulative index of issues 1-3, see the Developer Essentials *disc. A cumulative index of* develop *will be printed in issue 4 each year.*

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y,Z