The Developer Business Report May 1994

AppleDirections

Inside This Issue

Don't Miss This Year's Developers Conference

As we go to press, there's still room at this year's Worldwide Developers Conference, to be held May 15–20 at the San Jose Convention Center. You can register electronically by using a form posted on AppleLink (path—Developer Support:Developer Services:Events/Marcom:WWDC). Or, contact 1994 Apple Worldwide Developers Conference, CMI, 120 Montgomery Street, 5th Floor, San Francisco, CA 94104, U.S.A.; fax: 415-598-4301.

Apple News

Power Macintosh Tools Announced

The announcement of a full suite of Power Macintosh development tools by Apple Computer, Inc., and 13 tool vendors capped Power PC–related developments in the weeks immediately following the shipment of the first Power Macintosh computers.

Apple followed up on its March 14 worldwide introduction of the next generation, RISC-based Macintosh systems by announcing the third-party tools plans in addition to other news about the new computers, including these items:

- Apple released the next version of its Macintosh on RISC Software Developer's Kit (SDK).
- Power Macintosh computers outperformed Pentium-based PCs in tests conducted by Ingram Laboratories.

Additionally, IBM and Motorola announced a new version of its PowerPC 601 microprocessor, the chip that drives the first Power Macintosh computers, that will run at 100 MHz. Volume manufacturing of the 100 MHz PowerPC 601 is expected to begin in the fourth

please turn to page 8

Strategy Mosaic

Staying Bullish on Newton

By Gregg Williams, Apple Directions *staff*

I like getting letters because they let me know what developers are thinking. Some weeks ago, I got a letter that caught me by surprise because it flamed about what I thought was a pretty good column—my February 1994 column about last December's Newton Platform Development Conference. (For those of you not familiar with the term, "flaming" occurs when someone writes, usually by e-mail, to disagree rather emotionally with what the recipient of the flame said previously.)

The letter had some valid points, and I realized that other developers might be asking the same questions. So I thought I'd share my reply with you:

Pete,

*Congratulations on writing a truly *effective* flame—emotional enough for you to vent your emotions, but with enough information and poorly concealed politeness <gasp!> to form the basis for a true discussion. I hope that this reply will cause you to take another look at my article and your opinions of both it and the Newton platform.*

Much of your letter is summed up in your statement "How can I be sure this product is going to be around for a while? I am deeply concerned! The installed base is only 80,000, most of whom seem to be developers, the press

please turn to page 4

AppleDirections

Volume 2, Number 5

Apple Directions, the monthly developer newsletter of Apple Computer, Inc., communicates Apple's strategic, business, and technical directions to decision makers at development companies to help maximize their development dollar. It is published by the Developer Support Information group within Apple's Developer Press.

Editor
Paul Dreyfus (AppleLink: DREYFUS.P)

Technical Editor
Gregg Williams (GREGGW)

Business & Marketing Editor
Dee Kiamy (KIAMY)

Designer
Robert Stone

Production Editor
Lisa Ferdinandsen (LISAFERD)

Contributors
Juan Bettaglio, Pete Bickford, Alex Dosher, Tim Enwall, Amanda Hixson, Caroline Rose, Anne Szabla, Jessa Vartanian, Mark Vernon

Manager, Developer Press
Dennis Matthews

Manager, Developer Support Information
Greg Joswiak

Production Manager
Diane Wilcox

PrePress/Film
Aptos Post

Printer
Wolfer Printing Co., Inc., Los Angeles, CA

Editor's Note

Sailing With the Wind

The press has been saying a lot of positive things about Macintosh computing lately, both the computers and the software that runs on them. A nice change, at least from our perspective, and I'd imagine from yours. As Ian Diery, general manager and vice president of Apple's Personal Computer Division, said at the Power Macintosh introduction event, "The wind is at our backs."

You and I have known for a long time that the Macintosh platform and your products have great things going; it looks like some influential analysts and writers now agree, thanks in part to the introduction of Macintosh computers with PowerPC RISC microprocessors.

Take, for example, the latest *Microsystems Service* report from market research firm Computer Intelligence. In this report, the InfoCorp subsidiary's analysts conclude, "We expect the Apple Power Macintosh to be one of Apple's highest volume, most successful product families."

Then there's Bill Machrone's column in *PC Week* (yes, that's the trade publication for owners of PC/DOS/Windows machines) last month, in which he says, "My next computer will be a PowerPC."

He has a lot of positive things to say about Macintosh computing:

Apple appears to be a good six months ahead of IBM in terms of product development and software integration. It has worked out the important elements of software compatibility and legacy applications. Not only does the new generation of Power Macs run all the old 680x0 Macintosh programs, but through Insignia's SoftWindows, it runs most Windows and DOS applications, too.

I'm excited by the creativity going into the applications themselves. The sheer performance boost of the PowerPC frees designers to do things they would never have attempted on a desktop machine, such as real-time zooms and drags, rich 3-D interfaces, and software-based voice synthesis and recognition. In all cases, real time means real fast.

I also want a machine that's more tightly integrated than anything from the Intel camp. I'm tired of conflicting drivers, hardware interrupts, and programs that fall down and go boom just because I loaded a new app or a new piece of hardware.

Then, of course, there's *MacWEEK*'s February 28 lead story, "The 6100: Mac RISC Pays Off," in which three writers, none of them known to be shy about ripping new products (and Apple) to shreds, give the new Power Macintosh 6100/60 very high marks.

According to Rick LePage, Robert Hess, and Stephen Howard,

Last week, MacWEEK's news group completed its first compatibility and performance tests on a prerelease Power Macintosh 6100/60. The results of our testing—which was done without Apple's knowledge or cooperation—give ample credence to the company's claim that PowerPC-based Macs will offer very high compatibility with today's software. Although we examined the low-end model, in some areas Apple exceeded even the lofty expectations it has generated.

Despite being loaded with current extensions, control panels, and fonts written for existing 680x0 Macintosh computers, the computer *MacWeek* test-drove started up without problems. Of nearly 100 680x0-based Macintosh applications, only one crashed the Power Macintosh system. As far as performance went, the writers conclude that

The 60-MHz PowerPC most closely matched a PowerBook Duo 270c, which has a 33-MHz '030. However, in window scrolling and disk and memory access, the Power Mac's results rivaled 25-MHz and 33-MHz '040 systems.

To go along with all the positive Macintosh reports, I've also been noticing more negative reports about the competition. For example, in a March 1 story called "Befuddled PC Users Flood Help Lines, and No Question Seems to Be Too Basic," the *Wall Street Journal* reported that, although the manufacturers of IBM PC compatibles "are finally having great success selling PCs to households, they now have to deal with people to whom monitors and disk drives are as foreign as another language."

The story listed a variety of basic problems first-time users have with PCs. My favorite: "So many people have called [one leading manufacturer] to ask where the 'any' key is when 'Press Any Key' flashes on the screen that [the company] is considering changing the command to 'Press Return Key.' "

Of course, this is a problem Macintosh users might also encounter, but the point is that *not once* did the story mention that new customers have been plaguing Apple's customer support folks.

Another PC story, in the March 20 *San Jose Mercury News,* points out a problem with PC compatibles that Macintosh users won't have to deal with: Apparently, many machines that run DOS/Windows use a battery to run the CMOS (complementary metal-oxide semiconductor) chip that goes dead after a couple of years. (The Macintosh computer instead uses long-life lithium or self-charging batteries.) When a PC is turned off, the CMOS chip stores time, configuration information, and other data PCs need for starting up. When the battery runs out, the PC just won't start and may lose the vital data. When PC users try to boot up, they see a message telling them that the computer "cannot find configuration."

According to estimates reported in the story, some 3 million users who purchased their PCs in 1991 and 1992 might see that error message this year. If they do, they have to reset their machines with setup information after they get a new battery. Don't you think those are 3 million users who are going to consider switching to Power Macintosh computers, real soon?

As Ian Diery said, with more positive ink than Apple and the Macintosh computer have gotten in a long time, it feels like we're no longer having to fight a strong head wind just to stay in the same place. That doesn't mean we stop working hard; instead, it means that all the effort will let us cover more distance than ever before and greatly expand our marketshare.

Paul Dreyfus
Editor

IndustryWatch: News & Perspective

When One Door Shuts, Another Opens

By Amanda Hixson, Director of Licensing, InfoWorld Publishing Company

Well, by now I'm sure you've heard that it's Adobe über Aldus—that is, that Adobe has taken over Aldus—and that Novell is gobbling up WordPerfect while taking a big chunk out of Borland as well. Toss in the Electronic Arts mash with Brøderbund and things are looking pretty grim for small companies in traditional softwareland these days; merger mania and corporate buyouts seem to be *de rigueur.*

Unlike some prognosticators, I don't think this necessarily means that the end of the garage shop is drawing near and that you should all expect to be servants of Microsoft or some other corporate giant in the near future. Much like the publishing industry some years back, the software industry seems to be rushing together to the point where we can expect to see three or four huge planetary bodies—say Microsoft, Lotus, Novell, and possibly a dark horse like Computer Associates—emerge to roam the software heavens. Each of these companies will probably own or have alliances with a number of smaller publishers, much the way each major publishing firm holds its own niche publishing satellite companies in tight orbit.

Like the book publishing industry, in which startup publishers flare up and then burn out as they enter the atmospheres of their larger cousins, small software publishers will continue to emerge and be consumed as the giants spin through their orbits. The trick is to gauge the paths of the large bodies and avoid their quadrants of space.

Apple Directions On Line—June

The June issue of *Apple Directions* will be available on AppleLink on May 15. To view the June issue of *Apple Directions* on line, follow the AppleLink path Developer Support:Developer Services: Periodicals:Apple Directions:Apple Directions June 1994.

As difficult as that sounds, there are still new technologies emerging every day, and with them come opportunities for small companies. Multimedia is a great example of this, as more and more products emerge in this area, and as customer demand for them increases. (For data about the rapidly growing market for multimedia products, see this month's Market Research Monthly on page 20.)

Another area of potential growth is in on-line delivery of information. In my job as InfoWorld's director of licensing, I spend a lot of time on line and I see numerous developer opportunities in this arena. For example, we still don't have great tools for moving large blocks of graphic data quickly and economically over any existing online services. In those places where there is an ability to move information, such as on the Internet, there are some serious constraints resulting from trying to do things that work for the least-common denominator hardware configuration. People will definitely make money if they can figure out how to do more to deliver graphics and sound within the constraints of NCSA Mosaic, for example, to a broad range of users. (NCSA Mosaic is an interface that lets you point and click on hyperlinks to move easily around the Internet; it is available on the net or on discs that accompany several books about the Internet).

Additionally, I believe that we'll see the emergence of entirely new forms of software as we move to object-based operating systems and frameworks such as OpenDoc. Monolithic products—such as traditional word processors and spreadsheets—will no longer make sense. It is possible to make money selling component software and agents; also, with products such as QuickDraw GX, you'll be able to execute incredible features with a minimum of code.

Fahren on the InfoBahn

I now have an on/off-ramp to the InfoBahn in my living room. That's right, I now have an ISDN line, a bridge, and all the appropriate connections to connect to the Internet. And let me tell you that connecting to the Internet at 56 kilobits per second from my Macintosh Quadra 800 computer is way cool. So why am I telling you this? Simple. I think there are a number of opportunities for software developers and people with other types of software expertise on the Internet.

Another interesting thing about the Internet: People are always looking for, and broadcasting, information about software and software development. If you spend some time surfing the net, you're bound to locate interesting software ideas, as well as some potential programmers and software authors. If you haven't spent any time on the net, I'd suggest you give it a try.

Microsoft to Offer Decryption Over Telephone Lines

According to an article on the Dow Jones News Service, Japan's Nippon Telegraph and Telephone (NTT) and Microsoft are joining forces to add a new wrinkle to the delivery of software by CD-ROM. The new Microsoft and NTT scheme is different from some current schemes, such as that used by Apple's Software Dispatch group, where the customer calls a telephone number, gives a credit card number and then receives an unlocking code from an operator to unlock and install a particular software package from a CD-ROM. The Microsoft and NTT agreement allows users to have their computer call a telephone number that connects them to another computer and then download a decryption key by modem. Look for this service to be available in about a year.

This method seems to remove one of the highest costs in this type of delivery; the transaction cost resulting from having human operators. If it's successful, you can expect to see other distributors soon follow suit. ♣

Amanda Hixson, long-time Apple veteran, high-tech journalist, and industry analyst, is currently director of licensing for InfoWorld *magazine. She can be reached at A.HIXSON on AppleLink.*

Strategy Mosaic

Staying Bullish on Newton

continued from page 1

has canned it, and no one is currently buying it."

I have some other points to make, but the most important one causes me to point you to an article in Apple Direct *(as it used to be called), the May 1992 issue, page 9. (You can even find it on the March 1994 Developer CD, pathname Dev.CD Mar 94:Reference Library:Periodicals:Apple Directions:Apple Direct 1992, in the Business and Marketing folder for that issue.) The article, "Crossing the Chasm," is by Geoffrey Moore of Regis McKenna, Inc., and it's really worth your time.*

Mr. Moore's thesis is that every new technology that survives goes through three stages: the early market, the "chasm," and the mainstream market. The early market, of course, comes from "early adopters," the people who just love the technology so much that they have to have it.

Mr. Moore describes the period of the "chasm" as follows:

"During this time, the product has by now been on the market long enough for technology enthusiasts and visionaries to have heard about it and, if they are interested, to have bought it. But the product is not yet established as a 'safe' buy; it has not, in other words, secured a market-leadership position in any particular segment of the market. From the point of view of a pragmatist or a conservative, it is simply too soon to take a chance on the purchase. As a result, during the chasm phase, sales dip precipitously—often just at the point where you have promised your investors a surge in revenue."

Sound familiar?

What does it take to get across the chasm into the mainstream market, the place where Mr. Moore says "all high-tech wealth is generated"? Several things (which is why you should read the article). He sums up his ideas by saying that a technology can't navigate past the chasm until the "whole product" is there:

"The whole product is the complete solution, the entire set of products and services needed by users to achieve the value proposition promised by the product you are selling. Pragmatists evaluate products based on whether the whole product is a proven solution and one that is readily available. Until it is, they won't buy."

Now, from what some of the speakers at the Newton conference said (all this is in my February 1994 Strategy Mosaic), a wireless communications infrastructure will be part of the whole product, and Robert Growney of Motorola said that his company would deliver two-way wireless technology "in 1994." My piece also said that developers who had created Newton products estimated that doing so takes "3.5 to 5.5 months." Granted, your company isn't going to be doing communications products. But if that were a possibility for your company, you could do the arithmetic and conclude that, if you wanted to be in on the ground floor of this market, you'd better get started, soon.

*As you well know, everything in business revolves around a tradeoff between risk and profitability. The only way to *be* there when the Newton market really takes off is to start working on products before the market emerges. Sure, there's a risk there, but you must accept that if you want the *big* win. In short, you gamble. If you want a medium or small win, though, the right thing to do is to wait and see.*

*But, you say, my Strategy Mosaic gives no information on which to base a hunch that the Newton *is* worth taking a risk on. Frankly, I think you should read my article again. The quotes are from top people at companies like Motorola, BellSouth MobilComm, Sharp, and Monsanto. (Some consultants quoted were also staking their careers on what they said at that conference.) Don't you think their opinions are worth something? Don't you think that even the fact that they chose to speak at the conference means something? As I said in my Strategy Mosaic, this isn't just Apple telling you how great things are going to be, this is people from other companies saying the same thing—and remember that they're putting *their* time and money into Newton development, too.*

Here are some other thoughts related to the Newton and your concerns:

- *I'm sure you've considered the parallels to the Macintosh 128K. It took a *long* time to grow, but look where it is today. Some dissimilarities are also important: the Macintosh *didn't* have a developer conference with 700 paying customers five months after it was introduced, nor did it have several products shipping several months after introduction. The Newton did.*
- *The press has canned it, you say. Not totally true. The*

**early* press canned it because they wanted to get something out quickly and didn't have the time to come to understand the product. But when* Macworld *got around to actually *reviewing* the Newton MessagePad (December 93, page 52), their review was much fairer. (In fact, it ended with "Although Apple released the Newton about four months early, and marketing claims led to mistaken expectations about the details of handwriting recognition, the Newton MessagePad is an intelligent piece of work with an impressive variety of serious business uses.") Unfortunately, that review didn't make the cover of the magazine, and most people never saw it. But I do agree with you that Apple needs to do more to counter the bad press the Newton has gotten.*

*I hope *my* flame to you contains more light than heat, and I hope it may change your mind on some points. Though I (an avowed liberal) cringe at doing so, let me close by offering you some conservative advice: Stay the course. Let Newton be Newton.* ;-)

Best wishes,
Gregg Williams
Apple Directions

(FYI, Pete sent me a well-thought-out reply that agreed with some points but chided me for not talking about "the chasm" in my Strategy Mosaic last February. I decided he was right, and his comments led me to write this column.)

But wait—there's more! In the past few days, I've had conversations with Apple PIE (Personal Interactive Electronics) division management, engineers, Developer Technical Support, and testers. The information in the rest of this column is based on these conversations.

Adding to My Letter

Before I go on to other Newton-related topics, I'd like to add some things to what I said in my letter to Pete. First, I recently read another statistic that puts Newton MessagePad sales in a brighter light. In the Feb. 21, 1994 issue of *MacWEEK,* Richard Shaffer, principal analyst of Technologic Partners and a well-known mobile computing guru, said, "The PDA category sold 100,000 to 110,000 units the first year. Compared to the CD player—the hottest consumer-electronics product ever—which sold about 35,000 units the first year, PDAs look quite good." I don't mean to imply that PDAs will be the same kind of "horizontal" product that CD players are, but given that the Newton MessagePad has sold over 80,000 units and it hasn't even been a full year yet, I think it's premature—and even irresponsible—to call the Newton a failure. New technologies take time to gain widespread consumer acceptance, and the Newton platform is no exception.

The second thing I want to mention relates to my thoughts on the necessity of developing in advance of the wireless communications market. I recently asked a Newton engineer what developers should do to prepare for wireless technologies. His answer boils down to one sentence: *To prepare for wireless, develop for modem.*

The Newton Toolkit gives you access to a prototype called an *endpoint,* which Newton documentation defines as "the interface between the user and an underlying transport mechanism." The endpoint interface isolates the Newton programmer from the details of connecting with remote devices. This means you can create (and even market) a product that uses a modem to communicate with remote computers and devices. Then, when wireless communications become possible, you simply change the endpoint code, and your program is now a wireless application! So if you're thinking of getting into the wireless communications market, there's good reason to get started now.

Finally, one thing I should have mentioned in my letter to Pete is how good a development environment the Newton Toolkit is and how it complements the sophisticated Newton software architecture. Having programmed the Newton, my experience is that, compared to the Macintosh, the learning curve for Newton is about one-tenth as hard, you can do about ten times as much, and the process is several times more fun. Since you program your Newton application incrementally, you're always making visible progress. Your compile cycle can be as short as a few minutes, and you can create a commercial-quality application in a few months. The Newton Toolkit makes programming fun again.

Newton Markets

Probably the most important information I got from PIE management came from Philip Ivanier, manager of the PIE Development Relations Group at Apple. He said that you should definitely focus on providing business solutions. Apple is selling Newton MessagePad devices in record numbers (thousands of units at a time) to corporations who will be giving them to their sales and service forces. We're talking some big companies here, but I can't disclose their names because they consider their use of Newton devices a major competitive advantage over their competition. I can, however, say that the Monsanto Company has made an initial purchase of 2,000 Newton MessagePad devices to start a commercial service called Infielder (see text box on page 6 for details).

So what does that mean for you? It means several possibilities for making money. Most directly, if you have Newton programming expertise, you can do consulting or custom programming for these corporations, who will need custom applications for all the Newton devices they've bought.

Many companies will want to keep their development in-house. No problem—there'll be a small but profitable market for development tools, toolkits, and customizable frameworks that will help them get their applications developed as quickly as possible.

Finally, as more and more people are handed Newton devices, the more they'll be looking for shrink-wrapped "horizontal" business and productivity applications and enhancements. In time, with more Newton devices visible to the general public (and lower prices, better applications, and more capable Newton models), the consumer market will grow and become more important.

According to one PIE manager, many developers seem to be overlooking one particular strength of Newton devices: their extreme usefulness in gathering data in the field. A few Newton applications do so, and are very useful because of it. What do these applications do with such data? They connect to modems and phone lines and exchange data (in both directions) with databases on everything from personal computers to mainframes. (There's that communications angle again. . . .)

And don't forget anecdotal data—data that people in the field can collect just because they happen to have a Newton device with them. One company is using Newton devices to take customer orders—but the application that does so also allows the salesperson to note how much shelf space competing products are getting. Such information, gathered over an entire sales region, will help that company make better decisions.

The Newton Toolkit also allows you to transform text files into *digital books,* which Newton devices can display and search (among other things)—and creating digital books is even easier than creating Newton applications. Every profession has reference material that its practitioners would like to have close at hand, which means there's a vertical market for digital books.

Digital books of more general interest are also possible. What about an easy-to-use grammar book or dictionary of frequently misspelled words? A directory of consumer-electronics customer service phone numbers? A how-to manual for Adobe™ Photoshop or Microsoft Windows? Some enterprising developer could team up with a publisher and make quite a market for how-to books adapted to digital book format.

So let's say you have an idea for a Newton application but don't want to go it alone. (The smaller your company, the more likely this is.) You may want to consider contacting StarCore, PIE's publishing and distribution group. If StarCore decides to publish your product, it will take care of many of the details of publishing and pay you royalties based on sales. If you use StarCore to distribute your program, you take on more risks and responsibilities and may make more money by doing so. You can contact Sam Parker regarding publication or Ivy Millman regarding distribution. Both can be reached at StarCore, 5 Infinite Loop, MS 305-4A, Cupertino, CA 95014.

Product Ideas

When asked what would make a good Newton product, several PIE employees described something that would be the Newton equivalent of a FileMaker Pro. (It's an axiom of Newton development that trying to port a Macintosh application to the Newton ignores the differences between the two and will almost always result in a mediocre product.) As Bob Ebert, a PIE DTS engineer put it, "The application I envision will be sort of like, but entirely different from, FileMaker Pro. It'll be more of a data-collection, data-browsing, and forms-building application, with ties to a mainframe or desktop."

Given the suitability of Newton devices for data gathering, any application that will let users create customized databases (and the human interfaces for viewing and manipulating them) will be successful. End-users will want it, in-house developers will want it—you may even be able to create two different versions, each optimized for its intended audience. Different users will have different needs, so I can see the market supporting more than one such product.

Another useful product that came up in conversation was described as "something that stands between me and my Meeting Maker" (or other networked scheduling application). Such a product would have to include the ability to download your schedule from the server, modify it, and update the server. As with a database program, it must be incredibly easy to use—or people will not bother to use or buy it.

Another idea is an enhanced to-do list that works the way people work. It should take care of items that are partially completed—also items that, when completed, automatically generate additional items (tasks or reminders) for a future date. It might also alert the user regarding items that have a deadline or some other constraint attached to them. Such an application might even include some amount of critical-path project management in it.

Just about any vertical market is a source of Newton applications—the only limitation is whether or not you can make enough money from it to justify the effort. Take any profession that comes to mind—plumbers, car salespeople, grocery store managers, librarians, pilots—follow them around and take notes. You can probably find at least half a dozen things that, together, would make a good Newton application for them. If you can do something for them that is compelling enough, they'll buy a Newton MessagePad just to run your application. Years ago, people bought Apple II computers (which were far more expensive than Newton MessagePad devices) just to run VisiCalc spreadsheets.

And it bears saying—again—that you *shouldn't* think of

Infielder—Taking Newton Into the Field

The Monsanto Company has bought 2,000 Newton MessagePad devices to start a new service called the Infielder Crop Records System. Is this a real service? Well, I have in hand a four-color brochure and a promotional videotape for it, and any farmer with a 386DX IBM-compatible computer—yes, the very computer that the farmer already owns—can buy it for $999. This includes a Newton MessagePad, custom software for both the Newton MessagePad and the PC, and a 12-month subscription to various services from Monsanto.

Infielder allows farmers to use either their PC or a Newton MessagePad, whichever is handy, to keep track of the details of their farming. By keeping ongoing records of how their land is being used, what the weather is like, how much food each acre of land produces, what fertilizers have been used, and so on, farmers can begin to see exactly how well they're doing. (They enter much of this data on the Newton MessagePad while in the field—which they probably wouldn't do if they had to write the data down and type it back into the PC.)

Once they've dumped the raw data into the PC, the PC can then not only analyze the fields' productivity but also generate fertilizer- and pesticide-use reports required by the Environmental Protection Agency. In addition, farmers can connect to a "best crop management practices database" maintained by Monsanto, which allows them to compare their farming techniques and yields to those of other farmers working under similar conditions. By studying these data, farmers may be able to make changes that will increase the yields of their farms.

What does Monsanto get out of this? It gets closer to its customers, which is always a good thing. It also gets anecdotal data that indicates what chemicals farmers are using, how much, what for, and with what degree of success—which will help Monsanto streamline its operations and anticipate its customers' needs.

This is a good example of the exciting new business solutions that will be created around the Newton platform. Now imagine Newton devices in the hands of salespeople on the road, warehouse workers, doctors and healthcare professionals, stock traders, researchers, realtors, repair people working on site. . . . There are good developer opportunities out there: first, to get Newton devices into the hands of all these people by giving them custom solutions; and second, to sell them additional products that enhance the usefulness of the Newton devices they are already using.

porting your Macintosh or Windows application to the Newton platform. Instead, think about using the Newton to *enhance or extend* your application. A Newton device is portable, and it's good for gathering data. It's *not* a computer, but it complements one nicely. For more ideas in this vein, see my Strategy Mosaic articles on Newton in the September and October 1993 issues of *Apple Directions.* (They're also available on AppleLink and the March 1994 Developer CD.)

Real Soon Now

In the rest of this article, I'll be talking about matters of interest to programmers. If that's not your cup of tea, feel free to skip to the end.

By the time you read this, Newton Toolkit 1.0.1 should be available through APDA (U.S.: 800-282-2732, Canada: 800-637-0029; international: 716-871-6555). All registered owners of previous versions of the Newton Toolkit will be sent the final version free. Six months of intense use by developers has resulted in a faster, more stable version that has several very useful new features. Also, Newton Toolkit 1.0.1 ships with new, expanded documentation that folds the contents of many of the Q&A supplemental documents into the main documentation.

Also due Real Soon Now (some may already be out) are technical documents on using Newton devices to control devices remotely through infrared, multiple alarms (the Newton currently handles only one alarm), and optimizing your NewtonScript programs. They will be posted to AppleLink as Q&A documents, in the Newton support section that registered Newton Toolkit owners have access to.

Important Future Directions

I don't have any details about the feature sets of future Newton models, but several PIE engineers told me some of the things they're working on.

At last December's Newton Platform Development Conference, PIE engineers showed an experimental Newton system running compiled NewtonScript code. They recently said that this technology is "on track" but warned that a NewtonScript compiler is not a cure-all for sluggish NewtonScript code. Instead, they said, you can get better performance, *today,* by writing more efficient NewtonScript code. (The upcoming Q&A on that subject should help you out.)

The PIE engineers are looking at improving overall performance in two other ways. The first is by speeding up the Newton Toolbox and view system. A second is by improving the NewtonScript language interpreter. The engineers want you to know that they're working on improving the Newton platform based on developer and user feedback and that they've already implemented numerous improvements in the System Software 1.05 upgrade and the Newton MessagePad 100 and 110. This includes speed increases in application loading time, the view system, and the NewtonScript interpreter.

Just to make his point, Andy Stadler, a PIE engineer, used the Fodor's '94 Travel Manager to look up directions to a restaurant. Compared to a Newton MessagePad running system software version 1.04, he said, a Newton MessagePad 110 made the lookup approximately 30 percent faster.

Programming and Testing Notes

Here are some miscellaneous notes that you should know about:

• Before you release your Newton application, you should make sure it behaves itself if it's running from a PCMCIA card. In particular, the user should be able to remove the card without getting either the dreaded "Newton still needs the card you removed" message or a –10401 error. Michael Engber of PIE Developer Technical Support has written an article entitled "Newton Still Needs the Card You Removed," which was to appear in the February 1994 issue of *Double-Tap* magazine. Draft 5 of this article is also on the May 1994 Developer CD, pathname—Sample Code:Newton sample code 1.0:Errors:ART-NewtonStillNeeds-TheCard.

• Another very helpful article by Michael Engber is "Tales From the View System." It was published in the November 1993 issue of *PIE Developers* and is also on the May 1994 Developer CD, pathname—Sample Code:Newton sample code 1.0:Views:ART TalesFromThe-ViewSystem.

• Don't assume anything about the size of a Newton device's screen; instead, write your application to query the device for its size and adapt your application's layout intelligently. An AppleLink document on this subject is on the May 1994 Developer CD, pathname—Sample Code:Newton sample code 1.0:Views:resize link. PIE engineers say that your application should have both a minimum and a maximum size. Don't simply configure your application to take over the entire screen; on some future Newton devices, this may make your application look poorly designed.

• The Newton stylus can do more than just write. As one PIE engineer put it, "The Newton is a *pen* machine, not a handwriting machine." Use the interface elements appropriately. Use pop-up lists, radio-button clusters, and gauges when they make sense. Consider making use of the built-in gestures: tap, double-tap, scrub, highlight, caret, and line. (See the documentation on viewGestureScript in the *Newton Programmer's Guide* for more details.)

Catching the Big Wave

I'll reiterate what I said in my February Strategy Mosaic: "Even if the Newton platform were frozen today . . . we'd still have a pretty interesting market to exploit. . . . Since the Newton is also a new market with no companies in control of a particular niche, the Newton platform, *today,* gives any one or two people with a good idea and a modest amount of money a chance at building a successful business."

But the Newton platform *isn't* frozen. Already, improved models (the Newton MessagePad 100 and 110) have come out, and more Newton-based devices of various types are on the way from Apple and other companies. I've been hacking NewtonScript code since last October, so I know that one person can create a useful Newton application in weeks or even days, not (as with the Macintosh) quarters or—gulp!—years.

I'm unabashedly bullish on the Newton. But I also stress that going for the big win of being a leader in an entirely new market does entail some risk, and if you're not comfortable with that, you *should* wait and see. But it's like surfing—to catch the big wave, you have to be in place *before* it starts. ♣

Editor's note: My thanks go to the following people in the PIE division for their time and ideas: Philip Ivanier (manager, PIE Development Relations Group), Steve Strong (manager, Developer Information Group), Michael Brooks, Maurice Sharp, Kent Sandvik, Bob Ebert, Andy Stadler, Dave Temkin, Joseph Ansanelli, and Walter Smith.

Apple News

Power Macintosh Tools Announced

continued from page 1

quarter of calendar 1994. The rest of this story contains details about the announcements.

Power Macintosh Tools From Apple, 13 Other Vendors
You'll soon be able to purchase a wide range of development tools to help you develop native Power Macintosh applications. Native applications can take advantage of the increased processor speed of the new computers, which run two to four times faster than existing 68040- and 80486-based systems. The Macintosh on RISC SDK is currently available from APDA, which will also carry a number of other vendors' tools, as well. (For APDA ordering information, see page 24.)

The new products include native C and C++ development environments, a Macintosh implementation of Smalltalk, and a variety of tools designed to help developers make the transition to the Power Macintosh if they've never developed for the Macintosh platform.

The following companies have announced Power Macintosh development tools:

• Apple has been shipping a prerelease version of its Macintosh on RISC SDK since January. Now you can purchase a new beta version of the kit from APDA. If you purchase the beta kit, you'll also receive further interim releases up to and including the final release. Additionally, the kit will be included in the next issue of the APDA product E•T•O—Essentials, Tools, Objects.

• Absoft Corporation announced the availability by May of the Absoft FORTRAN 77 SDK for Power Macintosh and the Absoft C/C++ SDK for Power Macintosh. (For more information, contact Absoft Corporation, 2781 Bond Street, Rochester Hills, MI 48309; 313-853-0050.)

• ACI US announced the immediate availability of Object Master for Power Macintosh, an integrated programming environment for Pascal, C, and C++. (ACI US, Inc., 20883 Stevens Creek Boulevard, Cupertino, CA; 408-252-4444.)

• Bowers Development announced the immediate availability of its interface builder, AppMaker, Your Assistant Programmer for Power Macintosh. (Bowers Development, 97 Lowell Road, Concord, MA 01742; 508-369-8175.)

• Bare-Bones Software announced the availability by June of BBEdit for Power Macintosh, a programmer's editor. (Bare-Bones Software, 1 Larkspur Way #4, Natick, MA 01760; 508-651-3561.)

• AT&T Bell Laboratories announced the immediate availability of FlashPort Translation services for Power Macintosh. Using binary-to-binary translation technology, FlashPort translates existing 680x0 Macintosh applications into native Power Macintosh applications. (AT&T Bell Laboratories, Cruz Plaza, 943 Holmdel Road, Holmdel, NJ 07733; 908-946-1140.)

• Jasik Designs announced the immediate availability of the Debugger V2 & MacNosy, a high-level and low-level debugger for 680x0 and Power Macintosh applications. This product provides tools for code coverage analysis, incremental linking, and global disassembly. (Jasik Designs, 343 Trenton Way, Menlo Park, CA 94025; 415-322-1386.)

• Language Systems Corporation announced the availability by June of Language Systems FORTRAN/PPC and Language Systems Pascal. Language Systems FORTRAN/PPC is a FORTRAN compiler for native Power Macintosh applications. Language Systems Pascal is a Power Macintosh Object Pascal compiler. (Language Systems Corporation, 100 Carpenter Drive, Sterling, VA 20164; 800-2LANGSYS.)

• Metrowerks announced the immediate availability of CodeWarrior, a native Power Macintosh development environment for C, C++, and Pascal. In addition, CodeWarrior provides PowerPlant, an object-oriented application framework. (Metrowerks, 1500 Du College, Suite 300, St. Laurent, Quebec H4L5G6, Canada; 514-747-5999.)

• MicroAPL Ltd. announced the immediate availability of PortAsm, a 680x0-to-PowerPC assembly language translator. (MicroAPL Ltd., West Bank Techno Park, London SE16LN, United Kingdom; 447-1922-8866.)

• Prograph International announced the availability in the third quarter of 1994 of Prograph CPX for Power Macintosh, an application development environment featuring an application framework and application editors implemented in a visual, object-oriented language.

• Quasar Knowledge Systems announced the availability in the second quarter of 1994 of Smalltalk Agents for Power Macintosh, a tool for authoring applications and agents using Smalltalk. (Quasar Knowledge Systems, Inc., 9818 Parkwood Drive, Bethesda, MD 20814; 301-530-4853.)

• Sierra Software Innovations announced the availability by June of Inside Out II, a multi-user relational database engine for use with Pascal and C/C++. (Sierra Software Innovations, 923 Tahoe Blvd., Suite 102, Incline Village, NV 89451; 702-832-0300.)

• Symantec, as part of its announcement of Symantec C++ 7.0, announced the immediate availability of the Power Macintosh cross-development kit. The Power Macintosh cross-development kit allows developers to port their Symantec C++ applications to the Power Macintosh

Power Macintosh Beats Pentium on Price/Performance

Looking at the Ingram performance results reported in the adjoining news story together with estimated street prices of similarly configured models, the Power Macintosh computers emerge with a price/performance advantage. In other words, the Power Macintosh computers cost less than comparable Pentium-based PCs from Compaq and they have higher performance.

The following price/performance ratios were obtained by combining Ingram performance results with estimated street prices of each computer configured with 16 MB memory and a 500 MB disk drive.

Computer	Price/performance ratio
Power Macintosh 6100/60	2.28
Power Macintosh 8100/80	1.50
Compaq Deskpro 560 XE	1.05
Compaq Deskpro 566/M	0.93

platform. (Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014; 408-253-9600.)

PowerPC Beats Pentium

At the Power Macintosh introduction in New York's Lincoln Center, Apple Product Marketing Managers Jim Gable and Pierre Cesarini demonstrated that the PowerPC–based systems performed a variety of tasks faster than IBM-compatible PCs using Intel Pentium chips. Independent research conducted by Ingram Laboratories recently confirmed that Power Macintosh computers outperform the fastest Intel-based personal computers available today.

Ingram used the same applications on both Power Macintosh and Pentium-based Windows systems, measuring performance on 24 different tasks, including loading files, scrolling, spell checking, and applying filters. Among the study's key findings were the following:

- The 80 MHz Power Macintosh 8100/80 computer beat Compaq's top-of-the-line Deskpro 566/M Pentium-based computer by 60 percent. In some processor-intensive tasks the Power Macintosh 8100/80 outperformed the Compaq by over 300 percent.
- The 60 MHz Power Macintosh 6100/60 computer outperformed Pentium-based computers running at both 60 MHz and 66 MHz. The Power Macintosh 6100 beat Compaq's Pentium-based Deskpro 560 XE by over 30 percent. In a number of tasks, the Power Macintosh 6100/60 was over 10 times faster than an Intel 80486-based PC.
- Not only was the Power Macintosh 6100/60 computer much faster than the Compaq Deskpro 560 XE on the tasks measured, but it beats the Compaq model's price by more than $1,000.

Combining the results of Ingram's tests with estimated street prices, Power Macintosh systems currently enjoy a price/performance advantage over the Pentium systems used in the study. In fact, the Power Macintosh 6100/60 delivers more than twice the price/performance of the Pentium machines tested. For price/performance details, see the text box on page 8.

Faster PowerPC 601 Chip Unveiled

Apple will soon have the opportunity to extend the performance of Power Macintosh computers even further. IBM and Motorola, developers of the PowerPC RISC microprocessor along with Apple, recently announced 100 MHz version of the PowerPC 601 chip. Not only is the new chip faster than today's fastest (80 MHz) PowerPC 601; it is also about 40 percent smaller—about the size of a thumbtack. It also uses approximately half the power of current PowerPC 601 chips.

Limited quantities of the new chip will be available by this summer, with full-scale production beginning in the fourth quarter. Apple has not yet announced its plans to use the chip, although it is expected to use the full range of currently announced PowerPC microprocessors, including the PowerPC 603, PowerPC 604, and PowerPC 620, in forthcoming Macintosh systems. For details about the PowerPC chip family, see "PowerPC Processors—What's New" on page 33 of the April issue of *Apple Directions.*

GeoPort Gains Outside Support, Goes Cross-Platform

Apple plans to open its GeoPort technology to the PC and telephony industries. This will make GeoPort a cross-platform technology that will promote the growth of desktop communications and multimedia across a variety of computing devices, operating systems, and both analog and digital telephone lines. On March 2, 1994, Apple Computer, Inc., Aox Inc., and Analog Devices, Inc., announced their intention to provide cross-platform, plug-and-play connectivity between personal computers and telephones on corporate desktops.

"The combination of Aox's integration skills and Analog Devices' signal processing leadership provide an excellent channel to the installed base of corporate PCs," said Steve Manser, vice president of Macintosh Desktop Systems. "We look forward to working with partners throughout the PC, telephony, and silicon industries to take advantage of this cost-effective way to link our different products and solutions."

As a high-speed media communications interface, GeoPort can support voice, data, telephone control, audio, and video over any analog (POTS) or digital (PBX or ISDN) telephone lines to any desktop PC, workstation, or notebook computer. Apple has been shipping GeoPort for Macintosh personal computers since August 1993 and will include the technology in future PowerPC processor–based machines.

In a move to quickly facilitate GeoPort access throughout the industry, Aox plans to license GeoPort from Apple. Once licensed, Aox will then provide cross-platform GeoPort designs, development tools, and certification services to PC, PBX, and integrated circuit manufacturers.

Aox has agreed to make initial GeoPort implementations available on digital signal processors (DSPs) from Analog Devices and on DSPs for the PowerPC processor. Subsequent implementations will support other DSPs and other host processors such as the Intel 80x86. These implementations will also support industry-standard software environments, including Microsoft's Windows and IBM's OS/2. Analog will implement the GeoPort hardware interface within its line of codecs (compressor/decompressor chips) and its ADSP-2100 family of DSPs. These integrated circuits will, in turn, be used throughout the industry by sound-card, modem, and PC manufacturers.

The GeoPort announcements were made at the Intermedia Conference, held in San Jose, California, and at a special exhibition of information superhighway technologies hosted by Vice President Al Gore at the White House.

A major advantage of GeoPort is its ability to support any telephone line, including digital T1 lines. In addition, GeoPort has been designed to deliver isochronous, real-time streams of data at very low cost. This will allow computer and telephony suppliers to offer such features as

- high-quality, multiple-party video conferencing over the PBX
- document sharing with workgroups for collaborative computing
- fax/modem capability from any desktop PC, even if it is connected to a digital line, without the need for gateway services
- integrated telephone dialing, answering, caller ID, and voice and electronic messaging services from the desktop
- high-speed transfer of images and documents from scanners, digital cameras, and notebooks onto the desktop PC
- connection of high-performance V.32bis or V.34 modems to any PC without a throughput bottleneck at the serial communications port

Analog Devices predicted that the cross-platform availability of GeoPort will open the market for business audio, sound-card, fax/modem, and video functionality. "The cost of adding the silicon necessary to support GeoPort into a PC product will be insignificant at the systems level. With the tremendous benefits it will provide, virtually every desktop PC will be a target for GeoPort-enabled upgrades," predicted John Croteau, director of computer strategy and planning at Analog Devices.

New GeoPort-enabled products from PBX and PC manufacturers are expected to be introduced in 1994.

Apple Ships Macintosh Quadra 610 DOS Compatible Computer

DOS/Windows users now have two ways of running their software on Macintosh computers. They can use Power Macintosh computers running Insignia's SoftWindows software, which emulates an Intel 80286 chip at 80386 and 80486 speeds. Their other option is the Macintosh Quadra 610 DOS Compatible computer, which employs both a 25 MHz Motorola 68LC040 and a 25 MHz Intel 80486SX microprocessor.

The Macintosh Quadra 610 DOS Compatible computer shipped in March in the United States and selected international markets, along with a new card that employs both chips and upgrades Macintosh Quadra 610 and Macintosh Centris 610 computers to become DOS-compatible. The computer and card are designed to attract traditional DOS/Windows users to the Macintosh platform, broadening the market for your Macintosh products.

"By developing the most compatible personal computer, Apple intends to provide users with all of the advantages of the Macintosh platform while protecting their investment in both DOS and Windows-based software," said Ian Diery, executive vice president and general manager of Apple's personal computer division.

The Macintosh Quadra 610 DOS Compatible computer's dual processors let users work in Macintosh and DOS environments simultaneously. The processors work independently, allowing users to run Macintosh and DOS or Windows applications at the same time, cutting and pasting information between the two environments.

Announced at the Fall 1993 Comdex, where *Byte* magazine awarded it first place in its "Best New System" contest, the system has passed Microsoft's DOS and Windows hardware compatibility tests, and it will appear on Microsoft's Windows 3.1 Hardware Compatibility List. The computer comes preinstalled with MS-DOS 6.2 and ships with Apple's PC Exchange software. PC Exchange lets users manage their DOS and Windows files in the Macintosh environment just as they would Macintosh files, allowing users to open, copy, rename, delete, and save documents and folders.

Dual monitor support is designed to provide customers with the option of viewing the Macintosh and DOS environments at the same time, allowing the user to add a second display without purchasing an additional video card. The Macintosh Quadra 610 DOS Compatible computer supports most VGA, SVGA, and multisync monitors as well as the Apple 14-inch or 16-inch Macintosh Color Displays.

The same hard drive runs Macintosh, DOS, or Windows applications. Apple also offers an optional internal CD-ROM drive designed to run Macintosh, DOS, and Windows CD-ROM discs. In addition, DOS and Windows applications print to any Apple or Macintosh-compatible printer through a built-in serial port or optional Ethernet port.

U.S. Apple price for the Macintosh Quadra 610 DOS Compatible computer, which ships with 8 MB of memory and a 160 MB hard disk as well as on-board Ethernet, is $1579. The DOS Compatibility Card for Macintosh is available at the U.S. Apple Price of $399.

MAE Expands Your Market to UNIX Users

Apple CEO Michael Spindler has made it clear that Apple Computer, Inc., will soon license the Macintosh operating system to other major electronics firms, allowing your products to run on their platforms. In the meantime, Apple will ship a new product designed to bring the functionality and ease-of-use of the Macintosh operating system to users of Sun SPARCstations and Hewlett-Packard 9000 Series 700 workstations.

The product, called Macintosh Application Environment (MAE), will ship in the United States in late April and worldwide by the end of May. With MAE, workstation customers have access to the benefits of the Macintosh desktop, including the ability to run most Macintosh applications. Its release opens the market of millions of UNIX® workstation customers to your Macintosh products. By working with many of you, including Aldus, Attain, Claris, DeltaPoint, Deneba, Microsoft, Now Software, On Technology, Quark, and WordPerfect, Apple certified that hundreds of existing applications will work with MAE.

"The Macintosh Application Environment demonstrates Apple's commitment to making its unique technologies available to a wider range of computer users," said Morris Taradalsky, vice president and general manager of Apple Business Systems (ABS). "For a long time, UNIX customers have demanded access to high quality, low-cost productivity applications—MAE now satisfies this requirement."

MAE customers will be able to use the Macintosh graphical user interface through the Macintosh desktop and Finder, which will appear within a UNIX X window. Customers will have access to Macintosh System 7 features such as aliases, TrueType, Apple events, Balloon Help, QuickDraw, and 32-bit addressing.

MAE closely integrates Macintosh and UNIX, allowing customers to directly manipulate the UNIX file system from the Macintosh interface, cut and paste both text and graphics between X Window System™ and Macintosh applications, and administer UNIX systems through the Macintosh interface. The MAE architecture also supports workstation devices, allowing access to Macintosh-formatted floppies and CD-ROM discs from existing workstation drives.

MAE supports the Network File System (NFS), which allows users to access, display, and manipulate remote and local Macintosh, PC, and UNIX files. Apple plans to add support for AppleTalk to MAE in the future.

MAE runs on Solaris 2.3 or HP-UX 9.01 or later, and requires an X11 release 4 or later window display server. MAE is fully compatible with HP's Visual User Environment, SunSoft's OpenWindows, and OSF Motif.

The Macintosh Application Environment will sell for the U.S. Apple Price of $549. Pricing outside the U.S. may vary by country. ♣

Technology

CD Highlights

Tool Chest Edition, May 1994

I have several changes to tell you about this month, starting with the one you've probably already noticed: the cover.

We've been designing cover art around our whimsical CD titles, usually parodies of movie titles, for the last four years or so. We hope you've enjoyed them (or at least not groaned too loudly when you've opened up your monthly Developer Mailing). The Apple Computer, Inc., legal department, however, tells us that this long run has to end. Thus, our new look. Each edition (Tool Chest, Reference Library, and System Software) will have its own visual style and standard name: this month, we welcome Jeff Gunion, our artist for the Tool Chest editions.

You'll notice a few changes inside, as well. The new Contents Catalog has emerged from beta to the light of day at the top level of the CD; improvements over the old catalog include better search capabilities and the ability to open folders and files on the CD directly from the catalog. Give it a try and let us know what you think by completing this month's survey or by sending your comments to us at either DEV.CD on AppleLink or dev.cd@applelink.apple.com.

Finally, there have been some organizational changes. The Sample Code folder has been moved to this disc from the Reference Library edition, and there are two new folders within Tool Chest: Developer Utilities and Interfaces. Developer Utilities will contain general tools of interest to developers, such as ResEdit and the AppleLink application; new this month is SWAt, the tool we use to set the window sizes, locations, and scroll bar positions of the Finder windows on this disc. The Interfaces folder will provide a one-stop shopping experience for header files, including Universal interfaces, MPW interfaces, and aliases to many other sets of headers on the disc. If the headers you need don't appear on this month's disc, drop me a line and we'll do our best to include them next quarter.

Here's some of the new and revised material included on this month's installment in the monthly Developer CD series.

A/ROSE Version 1.2.1

This folder contains the A/ROSE version 1.2.1 system extension, including associated interface files and object modules. Also in the folder is the latest update for the A/ROSE software, version 1.2.1, required for Apple Macintosh coprocessor-based NuBus™ cards, such as the Ethernet NB, TokenTalk NB, and TokenRing 4/16 NB cards running on Power Macintosh computers. This release also addresses a compatibility issue with operations on Power Macintosh computers.

AE Sample Applications

This folder contains sample applications that demonstrate how to develop applications that support Apple events and the object model.

For example, 7Edit is a recordable, scriptable text editor. It includes sample C and Pascal code demonstrating how to develop recordable, scriptable applications using Apple events, the object model, and AppleScript. It also supports the Core and Text suites of events.

Apple Bug Reporter

Apple Bug Reporter version 1.5 is a HyperCard stack that allows you to report Macintosh software and hardware bugs. This is an update to Apple Bug Reporter version 1.0b9. It incorporates many bug fixes and has a cleaner interface. When you use it to file a bug report, the stack automatically puts the report in your AppleLink Out Basket. When you next log on to AppleLink, your bug report is sent to APPLE.BUGS.

Apple Style Guide 2/94

The February 1994 edition of the *Apple Publications Style Guide* provides developers the most up-to-date style conventions Apple uses in its documentation. Many new terms have been added to—and obsolete ones deleted from—this extensive revision. It includes general rules for style and usage; appendixes on technical notation, units of measure, writing help ballons, and creating glossaries; and a Newton style guide.

AppleGlot 2.1

AppleGlot version 2.1 is a text translation tool that succeeds AppleGlot versions 2.0 and 1.1. The new version is batch oriented while 1.1 was single-file oriented. New features include support for more Resorcerer template field types, new resource compare logic, and background processing.

BBEdit Lite 2.3.2

BBEdit Lite 2.3.2 is an evolution of BBEdit 2.2, the popular freeware text editor.

Note: This is *not* an Apple product. It is provided on an "as is" basis. Apple is not responsible for any problems you may encounter in its use.

Convert•Projects 1.0b2

Convert•Projects is a utility that will read a Think C or Think Pascal project and produce an equivalent (or near-equivalent) Code Warrior project. This is particularly useful for large

please turn to page 19

Human Interface

Speed Is an Interface Issue

By Pete Bickford

Many surveys have tried to determine what it is about a computer that makes its users happy. Time and time again, it turns out that the biggest factor in user satisfaction is *not* the computer's reliability, its compatibility with other platforms, the type of user interface it has, or even its price. What customers seem to want most is *speed.* When it comes to computers, users hate waiting more than they like anything else.

Having just revealed this startling fact, I can vividly imagine someone in the reading audience exclaiming, "That's great, Doc, but speed's a hardware thing. We don't design the machines, so there's not a lot we can do about it!" But a good software designer probably exercises more control over how fast a program executes than any hardware engineer. Even better, a good human interface designer can perform magic that makes a program *feel* like it's running much faster—even if its execution speed hasn't changed at all!

Real Speed and Perceived Speed

Computers actually have two types of speed: the benchmarkable real (machine) speed and the user's idea of how fast a machine is going, or its perceived speed. Of these two, the one that really matters is perceived speed. For instance, a 3-D rendering program that saves a few ticks by not displaying the results until the image is complete will inevitably be seen as slower than a program that lets the user watch the image as it develops. The reason is that while the latter program's users are watching an image form, the first program's users are staring impatiently at the clock noticing every long second tick by. Users will say that the first program ran slower simply because the wait was more painful.

If we want happy users, we need to maximize our programs' perceived speed. We can accomplish this task in three ways:

- by maximizing the real (machine) speed
- by doing the visible work first
- by "faking out" the user

These methods are not mutually exclusive—in fact, the best applications do all three.

Maximizing Real Speed

If you want to live like a millionaire, the easiest and most direct way is to start by getting yourself a million dollars. If you want your program to seem like it's running fast, the most obvious thing to do is to actually make it run fast. That is, maximize its real speed.

The good news is that the hardware folks keep finding ways to effectively double the processing power of our machines every couple of years, while keeping the price about the same. The Power Macintosh computer (which you've no doubt heard so much about recently) promises to accelerate this trend. The net effect of all this is that the available installed base is getting faster all the time without you having to do anything.

On the other hand, there's never been a machine so fast that the software folks couldn't think of some new technology to run that slows it right down again. The original Macintosh computer was a powerhouse for its time as far as hardware goes, but the demands of a WYSIWYG interface sometimes made it seem pokey when compared to character-based DOS systems.

Macintosh II users rejoiced at a computer that effectively ran at four times the speed of the Macintosh Plus, and then willingly gave up much of the potential speed increase to run in color. As machines continued to get more powerful, users added 24-bit video, file sharing, sound input and output, desktop movies, and so on. There's no real end to this trend in sight—nor is it a Macintosh-only phenomenon. Many have been the cries of PC power users whose clock-doubled monster towers were humbled by the demands of Windows.

In some cases, you can require that your software be run only on computers that possess a certain level of machine speed. Unfortunately, every machine you exclude in this manner means one less potential sale. Worse, some markets, such as education, are full of "hand-me-down" computers and upgrade much less rapidly than other markets. Requiring a fast processor or a floating-point unit in your software may banish you from these markets entirely. Heavy hardware requirements are generally bad karma unless you're selling into a very specialized or vertical market.

A better way to get machine speed is to carefully engineer your software's underlying algorithms. While hardware improvements can often double your execution speed, improvements in the way you access and manipulate program data can often speed your application by whole orders of magnitude.

One problem, in addition to all the technical challenges involved in this, is that programmers have to conquer the 2 A.M. urge to write "good enough" algorithms that only seem to run acceptably because they're being developed on the fastest systems available. When the same software is run on an average user's machine, all those "good enough" routines move as fast as a VW Bug trying to climb Mount Shasta. There's a school of thought that says programmers should be forced to work on the least-configured systems in their target market, instead of the high-end workstations they tend to use. While this seems a bit like cruel and unusual punishment to me, I've seen programmers who usually work with Macintosh Quadra 950 computers rewrite routines to work 20 times as fast when they were forced to run their own applications on a PowerBook 140 computer.

Do Visible Work First

The next challenge is to put your program's speed into the areas where it does the most good. In most cases, this means responding quickly to user input, bringing back initial results rapidly, and shifting as much work as possible to times when the user is busy doing something else.

Whenever you have a choice, do the work users can see first, then complete the rest while users are busy absorbing what they see onscreen. For instance, if your program has to get 10,000 rows of data from a mainframe, do everything in your power to bring back the first 40 rows of data and display them right away. Then, while users are looking over this data, you'll have time to start getting the other 9,960 rows. At the very least, you'll have provided users with reading material to help pass the time.

The converse of "Do visible work first" is that you can give *invisible* work low priority. A great example is the way file sharing starts up in System 7. On a large system, file sharing may take several minutes to scan the various disks and directories as part of its start-up process. The developers wisely made this a background process with a fairly small pull on the system's capacity. As a result, the user is able to get useful work done while file sharing completes its scan during the moments when the user is busy. No doubt file sharing could start up much more quickly if it took all the CPU priority and made the user stand by until it was finished. However, the real result of that would likely have been legions of unhappy users who would have preferred to disable file sharing rather than tolerate the delay.

Faking Out the User

All of the above is about preventing users from feeling like they're *waiting*. Waiting, as they say, is a Bad Thing. Luckily, researchers have recently made a major breakthrough in waiting science that has important implications for software design.

Glasnost has given the world access to the huge body of research from scientists in the former Soviet Union on the subject of waiting in lines. As it turns out, there are actually two distinct types of lines. A type 1 line typically stretches out of sight and hasn't moved since dawn. When the waiting customers arrive at the front of the line, they usually find bureaucrats providing halfhearted service while looking like they have better things to do. (Although the research here is based on toilet paper lines in the Soviet Union, a similar type of line-waiting culture can be observed at the San Jose Department of Motor Vehicles). Long-time residents of such lines are given to fits of resentment, depression, and sudden homicidal rage.

A type 2 line is marked by steady, incremental movement with diligent work on the part of the staff working at the front of the line. It turns out that if people suspect that they are in a type 2 line, they experience much less anxiety. Indeed, if the object at the end of the line is highly desirable (such as tickets to a Pink Floyd concert), they may even sense a sort of giddy anticipation.

If you must make users wait, it's important to give them the impression that they are waiting in a type 2 line. The keys are that users must know how long the wait is, see steady progress, and get the feeling that the system is working as hard as possible to make their wait short.

The chief tools we have for doing this on the Macintosh are the "busy" cursor and the progress bar. Busy cursors can say to the user, "I know you're there, your problem is important to me, and I'm devoting all my energies to your problem." The "I know you're there" part is especially important, since computers (and shop clerks) who just seem to "go away" without any warning while the user is waiting tend to find their reset buttons punched.

An animated busy cursor, such as a beach ball, can be especially effective in keeping users pacified. The trick is that for each turn of the cursor, users will infer that the computer did a certain amount of work. Thus, if you want to convince users that you're working very hard on their problem, spin the cursor a lot. As long as you don't go too crazy with this, you'll give users the impression that long waits must mean "I sure gave the system a lot of work to do" rather than "Gosh, that system is slow!"

There are two caveats here: First, don't make the animation look so involved that users begins to suspect you're wasting their time morphing icons rather than working on the real problem; and second, make sure that cursor movement is fairly steady. Since users will infer that the turn of a cursor means work is being done, non-spinning is taken to mean that the computer is doing something else (although in reality, exactly the opposite is usually the case).

Finally, progress bars are essential for any wait of 20 seconds or longer. A progress bar tells users where they stand, shows concrete movement, and gives them an indication of how long they have to wait. Remember, users who can see steady progress while waiting for an hour generally report less anxiety than users who have to wait with no feedback for fifty minutes.

Till next time,
Doc

AppleLink: THE.DOKTOR

Author's note: I'd like to thank Bill Fernandez and Bruce Tognazzini for many of the ideas on which this article is based; also "Dr. Bob" Glass, whom I first heard proclaim, "Speed is an interface issue."

Pete Bickford is a member of the Apple Business Systems human interface team.

Understanding the Power Macintosh Architecture, Part Two: A Skeleton Key to Mixed-Mode Issues

By Gregg Williams
Apple Directions *staff*

Learning to program in a new environment is often difficult because you're faced with the gap between "what" and "how"—you're given a large body of information in the form of documentation (the "what") and you have to translate that information into a plan for getting your task accomplished (the "how"). Often, a programmer's first steps involve a lot of trial-and-error programming, which is both costly and inefficient. The more "how" information you have, the more productive you're going to be. But there are two flavors of "how," and you will be the most productive when you know both how to get your task accomplished and how the underlying technology works.

Last month, I gave an overview of the Power Macintosh architecture, which will be sufficient for many readers. However, if you're going to be programming Power Macintosh computers or managing people who will be doing so (or if, like me, you're just fascinated by technology), you'll probably benefit from reading this article. The definitive word on how Power Macintosh software works is still *Inside Macintosh: PowerPC System Software,* and you should go there for details beyond the ones I give in this article. (That document is on the April 1994 Developer CD and is also available through APDA. See the first part of this article, in the April 1994 issue of *Apple Directions,* for more details.)

Be warned: This article is very technical. You may not be interested in the technical details. Even if you are, you will probably not understand everything on the first or even subsequent readings; believe me, *I* didn't.

The Implications of Mixed-Mode Software

In the past ten years, developers have written millions of lines of code for 680x0 Macintosh computers. Apple's commitment to backward compatibility dictated that the Power Macintosh computer run yesterday's, today's, and tomorrow's 680x0 Macintosh software. And it does, through its 68LC040 Emulator software, which allows the PowerPC processor to execute Motorola 68LC040 instructions.

For various technical and compatibility reasons, parts of Macintosh system software remain in 680x0 code, while others are in PowerPC code. This means that every Power Macintosh computer is switching, as needed, between executing 680x0 and PowerPC code. Consider some of the difficulties this presents:

• Existing 680x0 software has to run correctly without modification, even though some of the system software routines it calls are implemented in PowerPC code.

• The 680x0 and PowerPC processors pass arguments to routines in different ways; the Power Macintosh architecture must somehow accommodate these differences.

• System software routines currently implemented as 680x0 code may, in later Power Macintosh system software, be converted to PowerPC code. Both existing and new Macintosh software must always run correctly, even if the implementations of some system software routines change.

A new manager in system software, the Mixed Mode Manager, handles all this, with amazing success. Because of it and other parts of Power Macintosh system software, you can continue—in large part—to create Macintosh programs like you always have.

However, in a few situations you must program differently (or, if you're porting existing code, you must change your source code). To know what code needs to be changed, you must first understand how the Mixed Mode Manager handles switching between the 680x0 and PowerPC instruction sets.

Mixing Modes

When your program (either 680x0 or PowerPC code) calls a system software routine (in either 680x0 or PowerPC code), four possible combinations exist (as shown in "Execution of mixed-mode code" on page 15). Let's see how the Power Macintosh architecture handles each of them.

The easiest case is 680x0 code calling a 680x0 system software routine: The 68LC040 Emulator stays in control and emulates the trap dispatch mechanism you're already familiar with—that way, existing 680x0 code runs without modification. (For the purposes of this explanation, you can say that when a system software routine executes, it encounters a special A-line trap instruction that looks up the address of its code in a table called the *trap dispatch table* and begins executing the code at that address.)

The second case occurs when PowerPC code calls a system software routine that is implemented in 680x0 code. The PowerPC compiler/linker takes the source code and ties it to "glue" code (provided by Apple) that exists in a shared library. When executed, this glue code

- takes care of various housekeeping details
- determines the address of the 680x0 routine
- switches mode by executing the routine using CallUniversalProc (which I'll discuss later)

This may not mean much to you now, but the important thing to remember is that you write the same line of source code as you did for 680x0 Macintosh computers, and the Power Macintosh architecture and compiler "do the right thing." This glue code involves the Mixed Mode Manager, which executes the 680x0 routine (using the 68LC040 Emulator) and then allows the PowerPC processor to continue executing PowerPC instructions.

Enter the Routine Descriptor, Stage Left

The third case, 680x0 code calling a system software routine implemented in PowerPC code, is where things get really interesting. To describe this, I first need to explain the existence of a much-misunderstood part of the Power Macintosh software architecture, the routine descriptor.

Given that 680x0 code had to run without modification, the Power Macintosh engineers had to decide what to do with a system software routine written in PowerPC code. After all, they couldn't just let the 68LC040 Emulator plow through it, trying to execute it as if it were 680x0 code. They decided that, since no base of existing Power Macintosh code forced any compatibility limitations, they had a bit more slack in deciding how to solve this problem. Once they defined how Power Macintosh code must be written, everyone would have to follow their guidelines.

The answer that the Power Macintosh engineers came up with was to give certain routines an associated data structure called a *routine descriptor,* which gives the Mixed Mode Manager the information it needs to work with this routine (including a pointer to the actual routine). The beginning of the routine descriptor data structure is actually an A-line trap instruction. When the 68LC040 Emulator encounters this trap, it passes execution to the Mixed Mode Manager. The Mixed Mode Manager switches to PowerPC mode, lets the PowerPC routine execute, then returns control to the 68LC040 Emulator. With this solution, the value that needs to be in the trap dispatch table is not the routine's address but the *address of its routine descriptor.*

Enter the Universal Procedure Pointer, Stage Right

This sets the stage for the fourth and final code-calling/routine-called combination: PowerPC code calling a system software routine written in PowerPC code. You'd think this case would be as simple as that of 680x0 code calling a 680x0 system software routine. But *nooooo. . . .*

To accommodate the case of 680x0 code calling a system software routine written in PowerPC code, the Power Macintosh engineers had to put the address of the routine descriptor into the trap dispatch table. But guess what? In this last case (PowerPC/PowerPC), the PowerPC processor can't directly execute the routine descriptor.

This case is resolved in the same way as was the second case, described earlier. The PowerPC compiler/linker takes the source code and ties it to the same "glue" code that was described earlier. (This code is "smart" and knows what to do in each situation.) In this case, the glue code calls CallUniversalProc, which looks at the routine descriptor, sees that the code is PowerPC code, and calls it directly. Whew!

Okay, that's what actually happens. Now it's time to add one final detail, one that doesn't change anything I've just described but *simplifies* how you think about PowerPC software. (I'm told that many Power Macintosh developers have had trouble with this concept—I know *I* did—so understanding it now will save you a lot of grief later.)

Apple engineers named the values that are stored in the trap dispatch table *universal procedure pointers,* or *UPPs.* In general, when writing source code for Power Macintosh computers, you will use universal procedure pointers, not procedure pointers—and this fact is part of an overall design that minimizes the number of places you have to write code differently from pre–Power Macintosh days.

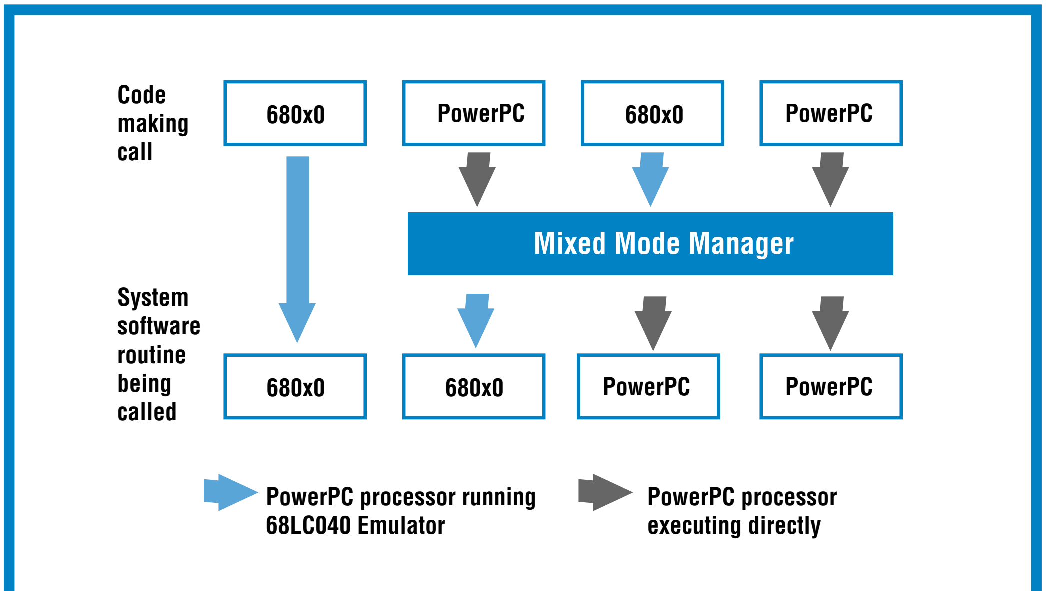
Read the next two sentences until you understand them (or until you have them memorized, whichever comes first). *If a routine is written in 680x0 code, its universal procedure pointer is a pointer to 680x0 code—that is, a pointer to the beginning of the routine. If a routine is written in PowerPC code, its universal procedure pointer is a pointer not to the beginning of the routine but to its routine descriptor.* (The figure "Universal procedure pointers" on page 17 illustrates this concept. As it turns out, a PowerPC routine descriptor points to something called a transition vector, which in turn points to the PowerPC code. For more information on the transition vector, see *Inside Macintosh: PowerPC System Software,* pages 1-26 and 2-5.)

Aside from being interesting in their own right, the explanations in this section introduce two key concepts, the routine descriptor and the universal procedure pointer. You need to understand both of these before you can answer the question "How do I know what code needs to be written differently for Power Macintosh computers?"

Dealing With External Code

If you were writing either 680x0 code that calls only 680x0 code or PowerPC code that calls only PowerPC code, you'd never have to worry about routine descriptors or universal procedure pointers. However, there are several situations in which your code must call *external code*—that is, code that is not directly contained in the code you are writing:

- You call a trap.
- You call a device driver.
- You use the address of a routine, but you don't know the instruction set of routine's code.
- You load and execute code from a resource.


Code making call
680x0
PowerPC
680x0
PowerPC
Mixed Mode Manager
System software routine being called
680x0
680x0
PowerPC
PowerPC
PowerPC processor running 68LC040 Emulator
PowerPC processor executing directly

Execution of mixed-mode code. In three cases out of four, the Mixed Mode Manager is involved when code from your program calls a system software routine in a Power Macintosh computer.

Power Macintosh system software takes care of the first two cases automatically. In the last two cases, you'll probably have to write your code differently from before, and that's what the rest of this article is about.

Passing Procedure Pointers

You can't write a Macintosh program from scratch without passing procedure pointers (that is, specifying a procedure by its address, then letting some other software routine call it using that address). Procedure pointers are used in several contexts:

- A handful of system calls use procedure pointers: for example, ModalDialog (which handles events when you display a modal dialog box) and TrackControl (which responds to mouse movement when the user has clicked a control).
- The fields of some records use them: for example, the ctrlAction field of a ControlRecord (a record that is associated with a button, checkbox, scroll bar, or other control).
- Some system global variables use them: for example, MBarHook (a routine that is called repeatedly while the mouse button is held down over a menu).

You need to be concerned about passing procedure pointers in two cases. The first occurs when you write source code that will be compiled to native PowerPC code. The second occurs when you write source code that will be compiled to 680x0 code.

Wait a minute! Didn't I say earlier that you should write programs the same as always if they're going to be compiled to 680x0 code? Strictly speaking, that's true. But Apple encourages you to write your source code as described in the sections that follow. It's not that much extra work, and doing so brings you considerably closer to having one source-code file that you can compile to run on either the 680x0 or PowerPC processor. Depending on your situation, making your source code "friendly" to both compilers may result in a larger market for your program.

Universal Header Files and ProcInfo Values

Like the pre–Power Macintosh header files, universal header files contain information about the Macintosh architecture that must be present when you compile your source code. Universal header files, however, contain additional information that allows you to use them regardless of whether you're compiling your software to 680x0 or PowerPC code.

You must use the universal header files if you are compiling to PowerPC code. You should also use them for existing 680x0-based software; doing so puts you one step closer to having source code you can compile to either processor. (See page 25 of *develop* issue 16 [December 1993] for details on how universal header files, also called *universal interface files,* differ from their predecessors.)

Before you can create a universal procedure pointer or call the routine it points to, you need to construct the routine's procInfo (short for *procedure information)* value. The main reason for the procInfo value is that the PowerPC processor passes parameters in one way, but 680x0 routines pass parameters in no fewer than five different ways (not including additional special cases). The procInfo value encodes essential information about a routine's calling conventions (and other characteristics). The Mixed Mode Manager uses this value to translate the passed parameters between the 680x0 and PowerPC architectures when a mixed-mode call occurs.

When you use the universal header files, you must change existing source code to use universal procedure pointers instead of procedure pointers. (The same is true if you're writing new source code.) The universal header files contain conditional statements that do different things based on whether you're compiling for the 680x0 or the PowerPC processor. For the former, these statements rewrite your source code to be the source code you would have written for a pre–Power Macintosh program. For the latter, these statements calculate the procInfo value for you and hide the mechanics of creating a universal procedure pointer and calling the routine from it. For the knock-down, drag-out details of exactly what happens see the file "Inside the Univ Hdr Files," located on AppleLink, pathname Developer Support:Developer Services:Periodicals:Apple Directions:Apple Directions May 1994.

(Note: The previous paragraph pertains to source code written in C or C++. You may have to code differently if you are writing PowerPC code in Pascal, FORTRAN, or some other language.)

Learning by Example

Let's look at an example of what you need to do to change a system software call that passes a procedure as one of its parameters. I'll use the system software routine ModalDialog, one parameter of which is a pointer to a procedure (call it myFilterProc) that handles certain events that occur while the modal dialog box is visible.

In existing, pre–Power Macintosh C code, you'd call this routine as follows:

```
ModalDialog
(myFilterProc, &itemHit);
```

To change this code to use a universal procedure pointer instead of the procedure pointer myFilterProc, here's what you do:

- Look in the Dialogs.h universal header file.
- Find the definition for ModalDialog; you'll find that it expects a parameter of type ModalFilterUPP. (The letters UPP remind you this data type represents a universal procedure pointer.)
- Find the macro that defines a new modal filter universal procedure pointer—in this case, it's called NewModalFilterProc.
- Add a line of code that uses NewModalFilterProc to convert myFilterProc to a universal procedure pointer.
- Finally, in the existing call to ModalDialog, substitute the new universal procedure pointer for the original procedure pointer, myFilterProc.

Your final code, which will compile correctly for both the 680x0 and PowerPC processors, will look like this:

```
MyFilterProcUPP =
NewModalFilterProc
(myFilterProc);

ModalDialog
(MyFilterProcUPP,
&itemHit);
```

The above code creates a routine descriptor, which takes up a non-relocatable block in the current heap. Since your application may create a lot of routine descriptors, you may want to dispose of them immediately. To do this, you'd add another line of code to the above code:

```
DisposeRoutineDescrip-
tor(MyFilterProcUPP);
```

Pages 2-21 and 2-22 of *Inside Macintosh: PowerPC System Software* describe this and an alternate way of dealing with deallocating routine descriptors.

In a similar vein, you can call the modal-dialog filter procedure itself by using another macro defined in Dialogs.h, CallModalFilterPointer, passing it MyFilterProcUPP. See the Dialogs.h file for details.

Converting Your Source Code to Universal Procedure Pointers

To generalize the above steps, here's how you change any system software call that passes a procedure pointer as a parameter:

- In the universal header file for the system software routine, find the corresponding new routine that changes the procedure pointer to a universal procedure pointer.
- In your code, use this routine to convert your procedure pointer to the appropriate universal procedure pointer.
- Call the system software routine using the universal procedure pointer you've just created (instead of the procedure pointer).

One of the things the universal header files do for you is to define the procInfo values for procedures whose parameters are known to Macintosh system software (in the example earlier, the modal-dialog filter procedure).

However, if you write a routine that passes a procedure that Macintosh system software doesn't know about, you must construct the procInfo value for the procedure yourself. (For details, see page 2-14, "Specifying Procedure Information," in *Inside Macintosh: PowerPC System Software.*) You then use this value when you create the procedure's universal procedure pointer (using the procedure NewRoutineDescriptor) and when you call the routine (using CallUniversalProc). To streamline this process, you may want to duplicate the functionality of the glue code in the universal header files that takes care of such details for you.

Executing Code From Resources

The second situation where you may have to change existing source code (or write new code differently) occurs when you load object code into memory from a resource and then execute it. Fortunately, because I've already covered most of the concepts you need to understand, this section is easier to explain.

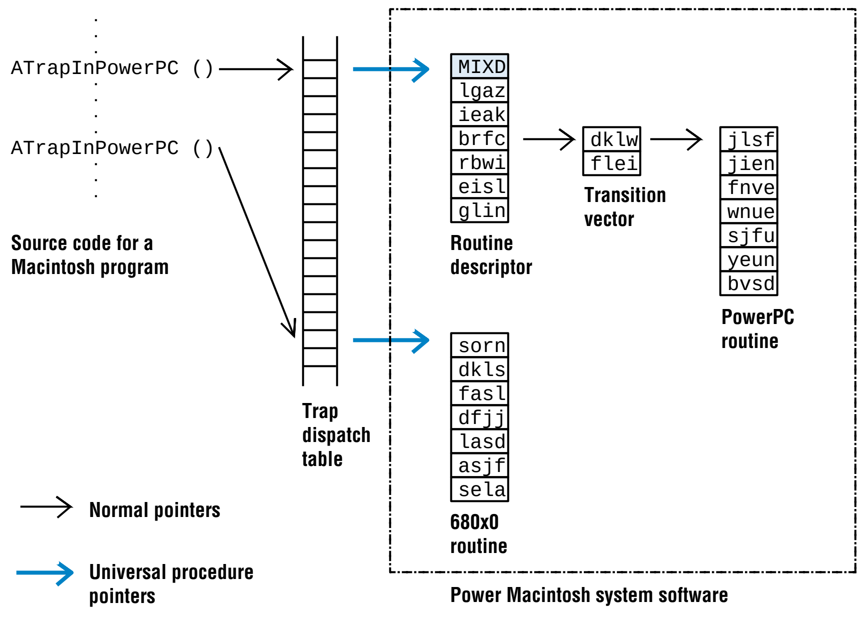
You have to look at the situation of executing code from resources from two different points of view: what you need to do if you're creating the resource, and what you need to do if your program is calling a resource.

Private and Accelerated Resources

If you're creating a resource that contains 680x0 code, you do it as you normally would. As I said earlier, existing 680x0 code—including code in resources—must continue to work as is.

Resources containing PowerPC code divide into two categories: *private resources* and *accelerated resources.* Private resources contain code that is used by your PowerPC program only. Because the conditions under which the code will be executed are known and because you have PowerPC code calling more PowerPC code, you don't need a routine descriptor, and calling this code doesn't involve the Mixed Mode Manager. Your program can simply load the resource into memory, use the Code Fragment Manager to prepare the fragment for execution, and execute it directly. However, Apple discourages the use of private resources; one reason is that they are not handled by file mapping (see last month's article for details) and so decrease the effectiveness of Power Macintosh virtual memory.

Accelerated resources are a different matter. An accelerated resource is any resource containing PowerPC code that has a single entry point and mirrors the behavior of a 680x0 resource. The great thing about an accelerated resource is that you can substitute one for its 680x0 equivalent, and existing code that uses that resource continues to work without modification. In some situations, replacing a 680x0 resource with an accelerated resource is a good way of speeding up existing software. Resources that you might consider accelerating include HyperCard XCMD



Universal procedure pointers. In Power Macintosh computers, universal procedure pointers (UPPs) replace the procedure pointers used in 680x0 Macintosh computers whenever use of the pointer might require a mode switch. A UPP for a PowerPC routine points to its routine descriptor (see text for details), but a UPP for a 680x0 routine is exactly the same procedure pointer as it would be in a 680x0 Macintosh computer. The shading indicates that the beginning of the routine descriptor is an A-line trap instruction that, when executed, transfers control to the Mixeed Mode Manager.

extensions and menu, control, window, and list definition procedures (which are stored in 'MDEF', 'CDEF', 'WDEF', and 'LDEF' resources, respectively).

However, accelerated resources, like private resources, are also not handled by file mapping. Because of this, you may want to package your routine as a PowerPC shared library, either appended to the application's data fork or placed in another file. In this way, your routine will benefit from file mapping.

Creating and Calling Resources

With the exception of private resources (which don't need routine descriptors), here are the rules to follow when you create executable resources (see also the table "Executing code from a resource," on this page):

• *For 680x0 resources, create them as you normally would in the 680x0 world.*

• *For PowerPC resources, add a routine descriptor to the head of the resource.* (To do this, use the Rez resource compiler utility and templates found in the MixedMode.r file.)

When you write code that loads a resource into memory and calls it, you should follow the following rules:

• *For 680x0 code calling a resource, you load the resource into memory and call it the way you always have.*

If the routine called is 680x0 code, the 68LC040 Emulator jumps to the beginning of the routine and starts executing it as 680x0 code. If the routine called is PowerPC code, the 68LC040 Emulator jumps to the beginning of it (which is a routine descriptor) and executes the A-line instruction that hands control over to the Mixed Mode Manager. The Mixed Mode Manager calls the Code Fragment Manager to prepare the code fragment for execution. Then the Mixed Mode Manager handles the execution of the PowerPC routine and returns control to the 68LC040 Emulator.

• *For PowerPC code calling a resource, you must load the resource into memory and then call it with the routine CallUniversalProc.* Note that CallUniversalProc requires a universal procedure pointer and its corresponding procInfo value. (As stated before, this excludes private resources, which need no universal procedure pointer.)

If the routine called is 680x0 code, CallUniversalProc invokes the Mixed Mode Manager, which causes the 68LC040 Emulator to execute the 680x0 code and then returns control to the PowerPC processor. If the routine called is PowerPC code, CallUniversalProc causes the Mixed Mode Manager to call the Code Fragment Manager (to prepare the fragment and return the entry point) and then to allow the PowerPC processor to begin executing the routine at its entry point.

Conclusions

Sometimes you have to know a lot to know what doesn't need doing. I think that's the case here; you have to know a lot about what goes on inside Power Macintosh computers before you can understand why there's not that much you need to do differently. Again, let me reemphasize that *Inside Macintosh: PowerPC System Software* is the one document that whoever's doing the programming needs to read and know thoroughly.

If you're making the decisions about porting existing code, writing new native (PowerPC processor–based) applications, or creating one set of source code that you can use to create both 680x0-based and PowerPC processor–based applications, all you need to know is that writing code for the PowerPC processor isn't all that different from what you (or your programmers) have been doing all along. Programmers' current skills aren't obsolete, and they don't need major retraining to make the transition to the PowerPC world.

Experience bears this conclusion out. Early PowerPC developers, who were working with prerelease development tools and minimal documentation, report that they took anywhere from a few days to a few weeks to convert existing C programs to run on Power Macintosh computers. One rule of thumb that I heard is to consider the conversion of an application to PowerPC to be roughly equivalent to a "0.5" revision—for example, from SurfWriter 2.0 to SurfWriter 2.5.

If you're developing from scratch, a native Power Macintosh application shouldn't be any harder to write than a traditional Macintosh application. In fact, numerous Power Macintosh features—like import libraries and global variables for non-application code fragments—make programming easier and should partially offset any other overhead that programming for Power Macintosh might add.

Of course, porting existing applications to Power Macintosh computers just keeps your foot in the door. People will be buying Power Macintosh computers in record numbers (see Pieter Hartsook's projected Power Macintosh sales figures in last month's issue, pages 29–36, for an independent analyst's estimates), and they will be hungry for applications that do new things that were impossible before the Power Macintosh computers arrived. Porting an existing application to Power Macintosh computers will give you enough breathing room for you to develop a competitive, native application.

Or look at it this way. The Macintosh Plus of the future (that is, the baseline, minimum computer that your software might

	...in 680x0 code	**...in PowerPC code**
When writing a resource...	Code as you normally would for 680x0 Macintosh computers	Add a routine descriptor to the resource
When writing the calling code...	Load it into memory using GetResource, then call the routine as you normally would for 680x0 Macintosh computers	Load it into memory using GetResource, then call the routine using CallUniversalProc (accelerated resources only)

Executing code from a resource. Often, you want to put executable code in a resource, then load it into memory and execute it. This table tells you how to create the resource and how to call it in the mixed instruction-set environment of Power Macintosh computers.

run on) will have a 60 MHz PowerPC 601 RISC processor, 8 MB of memory, built-in Ethernet, speech recognition and text-to-speech, the built-in ability to read and write DOS and Windows disks, and the numerous other features already standard on today's System 7 Macintosh computers. That's the *minimum.* So my question to you is: Do you have any idea what you can do with that? ♣

Editor's note: My thanks go to two Power Macintosh computer engineers, Alan Lillich and Mikey McDougall, who worked with me and reviewed both parts of this article. Mikey deserves special thanks for the considerable time he spent explaining the Mixed Mode Manager (again and again) until I fully understood it.

For a demonstration of the four combinations involved in calling and loading resources, try using the example program ModApp. You can find this program in the Macintosh on RISC Software Development Kit (available from APDA) and Metrowerks CodeWarrior Gold (Available from APDA or Metrowerks).

Also, the Graphing Calculator mentioned in part 1 of this article is located in the Apple Extras folder at the top level of the Power Macintosh hard disk (not in the Apple menu).

CD Highlights

continued from page 11

projects, in which manually adding and segmenting the project would be tiresome or impractical.

Convert•Projects is *not* a source code converter. If your code uses nonportable constructs that aren't supported by the Code Warrior compilers, you'll need to manually change your code.

Note: This is *not* an Apple product. It is provided on an "as is" basis. Apple is not responsible for any problems you may encounter in its use.

Developer Notes Update May 94

Included here are a new developer note describing the Macintosh DAV interface for NuBus expansion cards and a corrected version of the Power Macintosh developer note that first appeared in April 1994.

The *Macintosh DAV Interface for NuBus Expansion Cards* developer note describes the electrical interface for digital audio and video signals that Macintosh AV computers provide for NuBus expansion cards.

The *Power Macintosh Computers* developer note first appeared on the April Developer CD. The new version on this month's CD contains minor text and art changes.

Even More System Software

This is a temporary folder, containing some system software items we couldn't fit on the April 1994 Developer CD and others that have been lurking on the Tool Chest CD before being moved to the System Software CD. The updated contents of this folder will appear in their proper places on the July 1994 System Software CD.

Inside Macintosh: PowerPC Numerics

This book describes the floating-point numerics provided with the first release of the PowerPC processor–based Macintosh computers. It provides a description of the IEEE Standard 754 for floating-point arithmetic and shows how PowerPC numerics complies with it. The book also shows how to create floating-point values and how to perform operations on floating-point values in high-level languages.

Mac Application Environment

Macintosh Application Environment (MAE)—the virtual Macintosh for open systems—is an innovative software product for users of RISC-based UNIX workstations. This folder contains product information about MAE, as well as information about the MAE ISV Partnership Program to help Macintosh developers to penetrate the UNIX market with Macintosh applications. For more information on Macintosh Application Environment, see the news story on page 10.

Newton Sample Code 1.0

This folder contains Newton Q&A documents (in both DocViewer and Microsoft Word formats); sample code compatible with the Newton MessagePad, Newton MessagePad 100 and 110, and Sharp ExpertPad; and several articles on Newton development. Each Newton project includes a text file containing the project's source code (for curious programmers who don't have the Newton Toolkit).

ShowDialogBoxes Version 2.1

ShowDialogBoxes allows you to display an application's dialog boxes, alerts, and coolAlerts. It requires an associated script file that specifies the relationships between the text and the dialog boxes. Version 2.1 adds support for displaying coolAlerts under QuickDraw GX.

SWAt 2.0b5

SWAt is a tool that allows you to set several attributes of all Finder-related windows on a hard disk. The attributes currently configurable with SWAt are home window position, window stagger, uniform window dimensions, scroll bar position, and folder labels. This tool is designed to assist CD producers in presenting a uniform look to all windows in the Finder.

Verifier 1.0

The Verifier is an Apple internal localization verification tool. It is used to catch localization problems, such as corrupted CODE resources and mismatched resource attribute bits. It is customized to Apple's internal needs, but it may help you determine problems to look out for when you localize your software.

ZoneRanger 1.0.0

ZoneRanger is a freeware utility that provides detailed information about each heap zone that is active on the Macintosh computer. This information includes both the counts and total sizes of the free blocks, pointers, handles, locked handles, and resource handles in each heap zone.

Note: This is *not* an Apple product. It is provided on an "as is" basis. Apple is not responsible for any problems you may encounter in its use.

Next Month

A new artist, a new *Inside Macintosh* volume or two, and, perhaps, a new QuickDraw GX version.

Until then. . . .

Alex Dosher
Developer CD Leader

Business & Marketing

How to Order *The High-Tech Marketing Companion*

In recent issues we've told you about a new book, *The High-Tech Marketing Companion: Expert Advice on Marketing to Macintosh and Other PC Users.* In this book, by Dee Kiamy and the editors of *Apple Directions,* leading developers and industry experts describe practical techniques for solving business and marketing problems.

Due to popular demand, the book is now available by mail order from Computer Outfitters (formerly Mac's Place). To order, call 800-260-0009 (U.S and Canada) or 406-758-8000 (all other countries) and ask for *The High-Tech Marketing Companion,* product #7272. The price is $18.95 plus shipping and handling (plus tax if delivered to an Ohio address), a discount off the cover price. The book is also available at major bookstore chains in the United States and Canada.

Here's a sampling of what you can expect to read about in this book:

- how to avoid the ten most common product launch mistakes
- a step-by-step process for choosing and sticking with the best target market
- proven techniques for improving the number and kind of product reviews you get; how to deal with unfair reviews
- a systematic plan for choosing the right name for your product.
- developing packaging that helps your product stand out on the dealer's shelf
- creating demos that sell—when you can't be there
- techniques for getting and holding the attention of a national distributor

For a copy of the table of contents, call Dee Kiamy at Open Door Communications (408-266-9699) and leave your name and fax number, or send an AppleLink message to KIAMY. ♣

Market Research Monthly

Multimedia Market: Consumers to Take Center Stage

You just can't pick up a business publication these days without reading one of today's high-tech buzz words: *multimedia.* The concept of multimedia is neither radical nor new. It simply means the electronic publication of materials using several types of data, including text, graphics, animation, video, and sound, generally in CD-ROM format. The term, however, has taken on greater significance as more and more companies have jumped on the electronic publishing bandwagon.

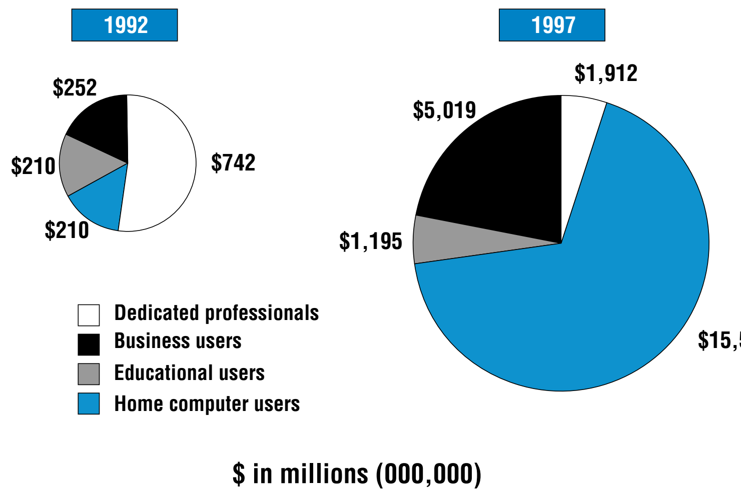
To help you understand the significance and future directions of this burgeoning market, this month Market Research Monthly provides data that originally appeared in the *CD-ROM Market Segmentation and Buyer Profile Report,* published earlier this year by Apple Computer, Inc.

The data, presented in the graph "Multimedia Market, Today and Tomorrow," suggests that if you're not developing multimedia products—either the devices for creating or reading content/titles, materials teaching others to create or read content/titles, or the content/titles themselves—you may want to think about entering the potentially huge market for them. Once you're in that market, you'll want to tailor your products to meet changes in customers' needs and, especially, to reach what's expected to be a gigantic consumer multimedia market.

According to the market research firm Market Vision, Inc., cited in the report, the worldwide market for multimedia products, including hardware and software, will grow to nearly $24 billion by 1997, up from $1.4 billion in 1992.

All segments of the market will undergo dramatic growth: Multimedia revenues in the so-called "dedicated professionals" segment—that is, people who use computers for design, print and other visual production, and visual arts—will

Multimedia Market, Today and Tomorrow



Source: Market Vision, Inc.

grow 157 percent. The general business market for multimedia products will be 18 times bigger in 1997 than it was in 1992, while the educational multimedia market will experience a four-fold increase.

The segment that will experience by far the largest amount of growth will be the home multimedia market, which is expected to increase nearly 72 times by 1997 from its 1992 size.

Expressed in more functionally related terms, by late in the decade, customers will be most likely to use multimedia technology for the following purposes:

- in business settings—for presentations, communications, and management-related tasks
- at home—for entertainment, instruction, obtaining information, and communicating

Many industry pundits point out that the much-discussed "information superhighway" may cut into the multimedia market, once the superhighway is in place. Much of the multimedia content available on today's CDs will be delivered over the information highway, or so its advocates predict. However, few think that the highway will be widely accessible to many—especially to consumers in the home market—before at least the end of the decade. The next several years, then, will present a ripe opportunity for you if you design the right consumer multimedia products.

Apple's *CD-ROM Market Segmentation and Buyer Profile Report,* which is full of valuable information about the multimedia market, is available to members of the Apple Multimedia Program (AMP) or as part of the Apple Multimedia Information Mailing through APDA. To join AMP, call 408-974-4897; APDA ordering information appears on page 24. ♣

Developer Outlook

Expanding Your Market Through a Developer Program

A Case Study of the ACI Experience

By Mark Vernon, ACI US, Inc.

[Editor's note: Even though you may not need to establish your own developer program, I urge you to read this article. Much of the advice and information ACI offers applies to creating good customer support programs in general, and to developing relationships with customer groups other than developers—such as user groups, VARs (value-added resellers), and other special-interest customers.]

We all have a variety of customer groups to attend to. And if you sell products that lend themselves to third-party development, you have a very special group of customers to work with. (Third parties are people who use your products to create other products of their own. To Apple Computer, Inc., they are fourth-parties.) These people are among the most valuable customers you can have, for reasons I'll discuss later. And you'll likewise need to give special thought to how you work with and treat them.

Even if you're a small company whose resources are already stretched thin, our experience is that it's especially important to contact and begin supporting this customer group as soon as possible. A great deal of ACI's success with 4th Dimension (also referred to as 4D) hinges on the relationships we've developed with our third parties—something we considered carefully from day one, when the company was quite small and not well known. Although we started small, we've built our developer program as the company has grown.

The Payoff

If our experience is any example, even though your company may not have the resources to manage a full-blown developer program, even a modest investment of time and effort can yield handsome rewards. Here are several strategic payoffs we've received from our investment in a developer program:

- *The program helps expand our market by multiplying the uses of our products.* When you look at 4D simply as an end-user database product, that's one mindset—one that could limit the perception and use of our product. But we (and our developers) don't perceive our applications as only database products; we perceive them as tools that developers use to create their own successful products. Naturally, this perception expands ACI's overall market.
- *Working closely with developers helps increase the quality of our products.* Our developers give us a tremendous amount of feedback about our products. Likewise, they are a large group of knowledgeable people who help us solve problems, test our products, and create new applications.

Furthermore, since our developers are the first people outside our company to receive beta versions of products, they supplement our efforts to find bugs and solve other problems before a product goes to market. This feedback, in turn, results in fewer user problems and more overall customer satisfaction.

- *Having an effective developer program builds customer confidence in our company and products.* Customers see how many products are based on 4D—and how many companies stake their livelihood on our application. Our reputation is closely tied to the reputations of our third parties, and vice versa. This interdependency creates a unique relationship that benefits both parties, and it significantly enhances the market's perception of ACI's stability and credibility.
- *The developer program creates an effective sales channel.* One of our developer programs that you'll read about later in this article is targeted at VARs. Approximately 20 percent of our total sales comes directly through VARs, who resell our products. These VARs have created their own applications based on 4D and related ACI software. Because our VARs are focused on selling the value of their unique, finished applications, ACI is able to reach new customers without creating conflict with our dealer channel.

We look at it this way: Our dealers make money by selling 4D as a general-purpose database system. VARs, on the other hand, create value-added products and then act as a sales force to create demand for *and* sell those products. The more products they sell, the more we sell.

• *The developer program helps us to be more effective in other sales channels.* We do sell our products through channels other than VARs, but historically we've tended to use other channels more as a fulfillment vehicle for the demand that ACI spends the money and other resources to create. Recently, however, after doing some analysis we selected approximately 200 key dealers in which to invest additional resources. Working with each dealer, we've either identified a well-chosen dealer employee to become an ACI developer, or teamed the dealer with one or more existing ACI developers. The benefit is that customers may find it attractive to purchase, from a dealer, a complete hardware/software solution that includes a customized 4D application.

• *The program creates differentiation.* Having a strong developer program can be a key differentiator in your overall competitive picture. If you demonstrate that you are committed to your product and markets and that you're in it for the long haul, you're more likely to attract developers to your product. This is especially important in highly competitive product categories. Developers stake their livelihoods on your plans and success; after they've made a substantial investment in learning your product and developing their own data and applications with it, they are reluctant to change.

• *The developer program leverages and multiplies our word-of-mouth efforts.* I believe that the single most potent way to sell software is by word of mouth—one person talking to another about a product. Word of mouth is especially important if you don't have resources for supporting a large advertising, direct mail, or PR budget. When we put our products into the hands of developers—advanced users, themselves—they become a source of recommendations and information to people who are making buying decisions. Therefore, our developer program creates a base of experts and influencers that significantly leverages and multiplies our efforts to spread the word about our products. And when those third-party companies generate word-of-mouth about their successful products that are based on our product, we reap some additional benefit.

How Our Program Is Structured

Today, we estimate there are 5,000 ACI developers worldwide, 3,200 of which are registered members of our developer programs around the world. However, in 1987 we started out very modestly in the United States with only a handful of developer contacts.

When you're a small company, any positive contact you have with developers—any support or information you give them, any perception that you create about your commitment to them—will go a long way. We started with a one-size-fits-all approach that included such things as offering developers advance beta versions of products and shipping them final versions before we put products on the market. We also began mailing developers technical tips and offering some small-scale training programs.

When we first started the program in 1987—for a few months before and after we first shipped 4D—developers who purchased 4D received our support for free. Soon after that we instituted a more formal program, which developers joined for a fee. (Today, it costs $895 to join one of our programs.)

These simple, first steps accomplished some important things. They helped make developers feel they were a valuable part of the ACI team. These steps also helped bring developers up to speed on our products; before a product hit the market, developers were already knowledgeable about it. This benefit was particularly important because developers represent themselves to their customers as being the source of information about products. Indeed, in many ways our developers *are* ACI.

As our company and developer base grew, we began adding more support elements, such as technical information mailings, training, and other things. Likewise, the resources we invest in this program have increased. Initially, one person in our company was responsible for overseeing the developer program. Now, three people do that job in the United States alone, supplemented by an outside firm that helps manage our annual developer conference. These three staffers work closely with other groups within the company, particularly our engineering department, which generates the technical information and support for developers.

Today, a significant portion of our marketing budget is allocated to our developer program. We support three kinds of developers: in-house corporate developers who create 4D-based products solely for use in their own companies; consultants, who create custom 4D databases for clients on a contract basis (systems integrators fit into this category); and VARs, who use 4D to create new applications for resale.

We've had to modify our initial approach to accommodate the different needs of each group. (There is, of course, some overlap.) We now offer two programs: The ACI Corporate Developers Program for in-house developers, and the ACI Professional Developers Program for consultants, which gives a developer the further option of becoming a VAR. (By choosing to become a VAR, a developer must also sign some legal agreements that dictate the terms of the resale relationship.)

While each program is tailored to its audience, the basic components of the programs are similar. (For a more complete description of what constitutes the ACI developer program, see the text box "The Anatomy of a Developer Support Program" on page 23.)

ACI Developers on the Global Front

Because ACI is an international company with subsidiaries in the United States, France, the United Kingdom, Germany, and Sweden and has distributors in many other countries, outside the United States we offer separate (but similar) developer programs.

Wherever possible, we tailor the content and language to the needs of the local market. Technical content information is coordinated between the United States and France for release throughout the world.

In countries where we have a distributor but no subsidiary, the distributor's developer program uses content generated by ACI. However, the distributor may add its own content to the program if our agreement allows this.

The Challenges

As I said earlier, the rewards from offering these programs are big, but running a developer program isn't without its share of challenges. Here are some things to seriously consider as you begin to put together a developer contact program of your own.

• *The program requires an up-front investment, and results may be difficult to quantify.* The money you put into your program may very well exceed the resulting revenue, and you may have difficulty quantifying the monetary benefits. Sometimes you'll be able to show a direct return on your investment; for example, the money we receive from VAR sales is easily tracked.

But what is difficult, if not impossible, to measure are the many intangible returns from our developer program investment, such as those I mentioned earlier (increased company credibility, word-of-mouth, and so forth). It's hard to measure the effect of these benefits on our overall sales, both short-term and long-term.

Finding ways to quantify and justify your investment is no simple task. While you know in your gut that your program is generating sales and multiplying your influence, attaching a number to it is difficult. Based on our experience, the best advice I can offer is to put processes in place to monitor your progress and ensure that you stay the course and that you're not spending money against some part of a program that isn't meeting your goals. Which brings me to the next challenge. . . .

• *Squeaky wheels can consume your resources and steer you off course.* As you well know, your developer group, like every customer constituency, will have more than its share of squeaky wheels. Be careful not to expend a disproportionate amount of your resources catering to them; there's a silent majority whose needs and issues can be very different from those of the more vocal crowd, and you need to be able to meet their needs, as well.

It's also very easy to fall into the trap of creating a strategy or changing your direction based on what you hear from the more vocal group. In fact, before you know it you could change your entire company strategy based mostly on input from squeaky wheels—a move that can prove highly damaging.

To stay on course, try not to let yourself become overwhelmed by this kind of input. Make the effort to reach out to the less vocal members of your developer group and learn what their needs and issues are.

Our "outreach" program includes a variety of activities. We invite developers to informal forums or meetings around the country. We also try to call all of our developers at least once a quarter. Because we have so many of them, this job is shared between our developer services team and the sales reps.

I also suggest that when a problem is brought up by a more vocal group, verify it with other contingents of your developer group before you respond to it.

The bottom line here is to manage your time and resources carefully, so that you can serve your overall group's needs and

The Anatomy of a Developer Support Program

While ACI offers two different programs tailored to the needs of its developers, the programs share some common elements:

• *Telephone technical support.* In the United States, for example, we previously had a single technical support phone number for customers and developers alike. Once developers identified themselves as being participants in our program, their calls were routed into a priority queue.

However, starting in May 1994 there will be a new toll-free support line dedicated exclusively to two groups: developers and users who purchase blocks of support time. Members of ACI's developer program will continue to receive free, unlimited telephone support and priority routing. (Users who don't purchase blocks of support time will have several other technical support options to choose from.)

A key consideration in structuring our phone support effort is making sure that the tech support operators who take developer calls are senior people who can answer the very technical and often complicated kinds of questions that developers ask.

• *Monthly mailings.* Each month, developers receive a package of information from ACI that helps keep them up-to-date about ACI and its developer community. The mailings' content varies each month, but generally it contains such items as technical notes, software updates, discount offers for ACI and its developers' products, product information about other developers' 4D products, and special offers from various hardware vendors.

• *Training programs.* Whenever possible, we conduct training classes prior to a new release to help developers prepare for when the product hits the market. While this training helps developers, it also gives us the bonus of getting final feedback from knowledgeable users, which we can use to fine-tune the product and class when needed.

Typically, these programs consist of a day or two of classroom instruction that includes hands-on experience with our software. Classes are held in Silicon Valley, in our New York offices, and also in rotating locations across the country on an ongoing basis. While all ACI classes are open to everyone, developers in our program get a first shot at "first-run" classes about new versions or products. They also pay a lower price for these classes.

• *An annual developers' conference.* At first this was a one-day conference, but over the years we've gradually expanded it in response to what developers say they need to accomplish. This year, for the first time the conference will span three days.

The main goal is to communicate to developers what ACI is doing at a strategic, corporate level: During the first day of the conference, we present our goals and plans for the next year and beyond. In addition, we conduct demonstrations of new versions and products that are under development. This prepares developers for what to expect from ACI and thereby helps them make better decisions for their own businesses.

Another important conference goal is to give attendees near-term information and guidance through seminars and workshops on a number of tracks. For example, we've conducted sessions on how to optimize database performance, design user interfaces, fully utilize add-on products such as 4D Write, and connect to SQL-based systems running on other platforms.

Non-members can attend our annual conference, but they pay more than program members do.

• *Support on electronic bulletin boards.* We maintain a forum on CompuServe and a bulletin board on the AppleLink network. The CompuServe forum contains a locked folder to which only our developers have access. It provides a place for developers to "chat" among themselves, exchange ideas, discuss problems, and the like. This forum has provided us with extremely valuable information. We also post utilities, tools, updates, and other software there for developers to download. In a similar way, the AppleLink bulletin board has also been helpful in supplementing our developer support efforts.

meet your program goals.

• *Accounting can be tricky.* Allocating expenses to and profits from a developer program can also be difficult. It's important to decide, at the beginning, where the money for the program will come from and where in your company the profits (or losses) will be posted. Getting agreement at the start from all affected parties or departments in your company is crucial to working peacefully with them when the program's bills start rolling in.

It all boils down to what your financial goals are for your program. Will you shoot for breaking even, or will you try to make money to invest back into the program? Will you view the program as a marketing expense, an engineering one, or a shared one? A cost center? A profit center? You should clearly answer these questions before launching your program.

At ACI, we've chosen to view our program as a marketing expense. Also, our revenues from VAR sales aren't counted as profits to the developer program but to the product lines themselves. It's easy to get buried in the accounting, and our marketing department spends a lot of time with the finance department to plan, budget, and track developer program expenses.

In the long run, our goal is for the program to pay for itself, and if we make a little bit of money, as sometimes we do, all the better: We then have additional dollars to invest into expanding the program.

• *Your mistakes can be amplified.* If you create this kind of developer community, when you make a mistake with a product you'll feel the backlash harder and faster. Our mistakes make an impact on developers' businesses, and such mistakes can potentially imperil our developers' livelihoods. But the fact of life is that with a complex product like 4D, mistakes are inevitable. We've done what almost all companies do at one time or another: failed to ship on a promised date, missed a bug—we're all familiar with the list of things that can go wrong.

However, the mistakes themselves aren't so bad; it's how you handle them and how you respond to developers that can make the difference. Knowing when and how to respond (or sometimes how not to respond) takes experience and insight. It can be a difficult call; often, the natural inclination is to pull back and lock the door. Try to suppress that impulse, even when the going gets tough. Avoid creating the "us versus them" mentality. Remember that the success or failure of these developers reflects directly back on you.

So try to keep the communication lines open; let your developers know that you understand the problem and the position it puts them in, and that together you'll find a solution. Tell them early and often, and repeat it until you resolve the problem.

At this point the squeaky-wheel dilemma really kicks in. You must quickly investigate each problem to determine the extent of its influence: Is it something that cuts broadly across your developer base, or is it only a problem that applies to the developers who brought it to your attention? Based on what you learn, act accordingly. For example, a broad-based problem may result in our distributing an update of our software. On the other hand, we may provide a "workaround" solution if the problem is limited to very few developers, and correct it in our next product release.

• *Enforcing quality control is difficult.* Because our reputation is closely tied to products our developers produce with 4D, it's important to closely watch the quality of those products. At first we created a Certified Developers Program, in which we reviewed developers' products. After a short while, however, we stopped doing that; it consumed a lot of time and resources because we found ourselves struggling to set up and enforce a body of subjective criteria. And in some cases, we found we had less expertise in a specific application area than the developer did. In the end we didn't net much from the effort.

We began to feel that customer satisfaction was a better indicator of how developers are doing. In most cases, if a developer's product isn't up to par, we'll hear about it from unhappy customers.

So we've shifted our focus away from setting up "certification hurdles" to constantly monitoring customer problems. When the occasional problem does crop up, we work with the developer to improve that product, through training, more technical support, or whatever else might be needed. Also, now that our company has a sizable sales force, we can be in direct contact with developers more frequently, and we can even visit user groups to get a better feeling about what's happening with customers.

How to Begin Your Approach

There's no real cookbook approach to starting a developer program. It's more like the menu in a Chinese restaurant—you know, the "family dinner" option that gives you choices depending on the tastes of those at the dinner table: pick one from column A, two from column B, and so forth.

The best advice I can give you is to size up your developers and their needs and determine what your own goals are. Then, take a good look at your pocketbook and choose from the many options at your disposal. If you find that you must experiment a little, as we have, don't cringe. A developer program is a living, breathing entity that changes as your company changes. Even for us, who have been doing these programs for a while, the picture is constantly changing. For example, as ACI moves its products onto several new platforms, we'll have to find ways to support new developers with different needs.

Remember, this is a not a destination, but a journey . . . so enjoy the trip! ♣

Mark Vernon is vice president of sales and marketing for ACI, an international company that develops multiplatform database products and whose U.S. headquarters are located in Cupertino, California.

APDA Ordering Information

To place an APDA order from within the United States, contact APDA at 800-282-2732; in Canada, call 800-637-0029. For those who need to call the United States APDA office from abroad, the number is 716-871-6555. You can also reach us by AppleLink; the address is APDA. If you're outside the United States, you may prefer to work with your local APDA contact. For a list of non-U.S. APDA contacts, see the "International APDA Programs" page in the *APDA Tools Catalog.*