The Developer Business Report *January 1996*

AppleDirections

Inside This Issue

Macworld Developer Central
Come see the latest developer products from Apple Computer, Inc., at Developer Central, part of this year's San Francisco Macworld Expo, January 9–12. We'll be giving away free copies of *Apple Directions* and *develop*, the Apple Technical Journal, and running demos of our latest tools and technology, so come on by!

Apple News

Apple's COMDEX Windfall

Company Showcases Variety of New Technologies

Buoyed by its recent market share gains, Apple Computer, Inc., took full advantage of the COMDEX electronics show—the largest U.S. computer trade show, held in November in Las Vegas—to showcase a wide variety of new products and technologies.

The announcements demonstrate the momentum Apple has attained coming off its record-setting fiscal year 1995 fourth quarter, or Q4 '95 (July through September 1995). During the quarter, Apple posted unprecedented revenues of $3 billion and sold 1.25 million Macintosh systems. Macintosh shipments worldwide grew 19 percent faster than overall personal computer shipments in the quarter, according to International Data Corporation and Apple data.

Another research firm, Dataquest, said that Apple sold 9 percent of all personal computers shipped worldwide during Q4 '95, up from 7.4 percent the quarter before, to reclaim second place in personal computer sales. Compaq gained the top spot in the Dataquest study, with 10 percent of worldwide shipments.

Among Apple's COMDEX announcements and demonstrations were the following:

- the PowerPC Platform specification,

please turn to page 11

Strategy Mosaic

International Development From Here to Copland

The World on a String (of Text, that is)

Amid the unfolding of the more dramatic chapters of the Copland story, a quieter but equally important part of the tale has fallen into place. I'm talking about the Copland solutions for international development, technologies that let you develop a single, globalized version of your software that can readily be localized for all the world's major computer markets. Apple Computer, Inc., calls this kind of software "world-ready," because it's ready to be localized and sold in any corner of the world where Apple does business, and not just in a single country.

In this month's column, I'm going to tell you about the Mac OS international technologies, in their current form and under Copland, because one of the nicer parts of the Copland story is that software developed using today's international technologies will work under the forthcoming next-generation version of the Mac OS. I'm also going to say why Apple's international technology solutions present significant opportunities for Macintosh developers, how you can begin to take advantage of

please turn to page 4

AppleDirections

Volume 4, Number 1

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 Apple Developer Periodicals group within Apple's Developer Press.

Editor
Paul Dreyfus (AppleLink: DREYFUS.P)

Technical Editor
Gregg Williams (GREGGW)

Business & Marketing Editor
Kris Newby (NEWBY.K)

Associate Editor
Anne Szabla (SZABLA)

Production Editor
Lisa Ferdinandsen (LISAFERD)

Contributors
Peter Bickford, Dave Curbow, Alex Dosher, Victor J. Hnyp, Ray Kaupp, Kerry Ortega, John Pugh

Manager, Developer Press
Dennis Matthews

Manager, Apple Developer Periodicals
Mark Bloomquist

Production Manager
Diane Wilcox

Prep and Print
Consolidated Publications, Inc., Sunnyvale, CA

Editor's Note

The Year That Was

What a difference a year makes. This time last year, as the experts peered into their crystal balls for 1995, Apple Computer, Inc., seemed shrouded in a dark mist. Apple's small market share was destined to evaporate. Developers were fleeing the platform, or so the rumors went. And it was going to be the year that the next version of Windows—the OS that was to be called *Windows 95*—finally buried Apple once and for all.

Right.

The reality of 1995 couldn't have been more different than the experts' predictions. I'll just step aside from all the editorializing and let the facts speak:

• Apple registered worldwide market share gains. In its fourth 1995 fiscal quarter (Q4 '95)—that is, July through September—worldwide Macintosh shipments grew 19 percent faster than shipments of all personal computers combined, according to International Data Corporation (IDC) and Apple's own data. During the same quarter, Dataquest said that Apple sold 9 percent of all personal computers shipped around the world, up from 7.4 percent the previous quarter.

• Apple also gained share in the United States. In Q4 '95, IDC and Dataquest said that Apple sold 13 percent of all computers shipped there, more than any other personal computer vendor.

• The Macintosh share of the world's number two market, Japan, continued to climb, growing from just over 16 percent at the beginning of the year to nearly 20 percent, according to IDC Japan and Dataquest Japan.

• Apple took a commanding lead in the U.S. educational market; according to Quality Education Data, Apple has a 63 percent share of the U.S. public school personal computer market, up from 59 percent at the end of 1994.

• In some specialized segments of the market, Apple maintains a strong, even dominant presence, to wit:

—the Macintosh computer holds a 76 percent share of the professional color-prepress market, a share that will grow to over 77 percent next year, according to Griffin Dix Research.

—Thirty percent of U.S. desktop publishing software revenues—and 24 percent of U.S. home education software sales—went to Macintosh developers in the first half of 1995, according to Software Publishers Association.

So much for the idea that Apple's market-share is evaporating. Now, what about the quaint notion that developers are fleeing the platform?

• A record number of developers—more than 3,500—attended the 1995 Worldwide Developers Conference, held last May. Half of them were attending the event for the first time.

• As of mid-October, developers had delivered more than 1,400 32-bit applications that run in native mode on Power Macintosh RISC-based systems, a 300 percent increase since the beginning of the year.

• Within three weeks of the posting of version 1.0 of the OpenDoc software development kit (SDK) at Apple's Web site, nearly 30,000 different people had initiated 48,000 requests to download documentation and software from the SDK; over 300 developers have already committed to delivering OpenDoc software.

• In Apple's spring 1995 survey of Macintosh developers, 97 percent said they planned to continue Macintosh development for the next two to three years, while 85 percent had plans to develop new Macintosh products.

Need I say more? Now, about the Windows 95 *coup de grace*—

• A recent IDC study concluded that the Macintosh software market is more profitable than the Windows market, and will grow at least as fast or faster over the next five years. (See "IDC Report Shows Mac OS Better Business Proposition" from last month's issue for details.)

• Dataquest recently downgraded the number of copies of Windows 95 it expects Microsoft to sell by the end of 1995 from 30 to 16.4 million units.

'Nuff said about 1995. Based on Apple's performance last year, I personally can't wait to see what 1996 brings. Happy New Year!

Paul Dreyfus
Editor

AppleWatch

Working With Apple: Three Perspectives

[Editor's note: On these pages we usually look outside Apple for news and perspective that affects the Apple developer community. This month, IndustryWatch steps aside temporarily to make room for a look inside Apple, which we hope will be useful to you.]

When Apple Computer, Inc., created Apple Developer Relations a few months ago, the most important reason for the reorganization was to make it easier for developers to work with Apple. Uniting Developer Technical Support (DTS), Evangelism, Developer Press, Developer University, Developer Programs, and other developer-related functions lets you work with a single, unified organization, and (we hope!) makes life better for you.

In that spirit, we've pulled together three views of how you can deal with Apple in a way that gets you the most for your efforts. The first two come from DTS engineers Dave Polaschek and Brian Bechtel and relate mostly to getting technical support from DTS. The third comes from new Apple Fellow Guy Kawasaki. He presents good general advice for working with Apple, whether or not you're a program member.

By the way, if you're not yet a program member, perhaps the best first step you can take in working with Apple is to join the program that's right for you. For more information on the programs and how to join, visit the Apple World Wide Web site (http://dev.info.apple.com/developerprograms.html), send inquiries to devsupport@applelink.apple.com (DEVSUPPORT if you use AppleLink), or call the Developer Hotline at 408-974-4897.

From Dave Polaschek

As an outside developer until recently, and a DTS engineer within Apple now, here are a few things I've learned about how to deal with Apple.

• *There's a lot of information available—use it.* While I don't want to discourage people from contacting DTS, it's going to be quicker and more productive for you to look information up first—in *Inside Macintosh,* at our Web or FTP site, in Technotes, and so forth, than to send us an e-mail message just to have us look it up for you. Save DTS for the really tough questions that can't be readily answered by available documentation.

• *When you find someone within Apple who can help, don't abuse that relationship.* When you get stuck on a really hard problem, having someone inside Apple who can give you some kind of answer is a very valuable, and limited, resource. Remember that your contact person has a job to do, too, and if you use that person's time answering questions about things you could've looked up yourself, or with talk of the weather, he or she may not have the time or inclination to help you when you really need help.

• *When you're corresponding with Apple, make sure you identify yourself.* This is especially important when working with DTS, but it applies to all correspondence with Apple. If Apple employees aren't sure who you are, they often have to take extra time to determine what they can tell you. When you send us e-mail, please include a return address (not all mailers automatically generate good return addresses), and a phone number (in case we can't get through via e-mail). All correspondence to DTS is automatically acknowledged. If you don't receive this acknowledgement, it means either that we didn't get your mail, or that we're unable to return mail to you. Remember that the Internet does not have guaranteed mail delivery.

• *When dealing with Apple, state your problem clearly.* I've been on both sides of the "DTS answered a question, but it wasn't the one I was asking" problem. It almost always results from asking a question and not giving enough background data for DTS to know what you're really asking. Taking an extra half-hour to figure out what your problem and your goals are before you ask your question, and stating it clearly, can save time in getting an answer.

From Brian Bechtel

Here are some thoughts on how to deal with Apple from the outside.

• Use e-mail, not the telephone, and try the official channels first. If this approach doesn't work, go around the channels, but complain, too.

• Put a useful subject line on the e-mail, not something like "technical question."

• Include your name and telephone number on all e-mail messages, so we can call you.

• State what you've already done, read, or coded in your attempts to solve the problem, so you won't get frustrated by answers suggesting what seems to you to be obvious.

• Provide sample code in your e-mail, if possible and appropriate.

• Tell us what extra conditions are involved (for example, "only fails on a Macintosh Quadra," "only interested in the education market," and so on).

• Give feedback to devfeedback@applelink.apple.com. This e-mail address is read by the Developer Relations managers up to and including the vice president. If we don't know something is wrong, we can't fix it.

Here are additional observations based on too many years at Apple:

• Using DTS is usually more efficient than calling random engineers for technical questions.

• If DTS doesn't help, don't stop using us; complain to devfeedback@applelink.apple.com.

• Having contacts in engineering departments works only if you continue to cultivate new contacts, since projects end, and engineers move on to other things.

January *Apple Directions* Online

January's *Apple Directions* will be available by December 15 at the following locations:

AppleLink: path—Developer Support:Developer Services:Periodicals:Apple Directions.

Internet: http://dev.info.apple.com/appledirections/adtoc.html

eWorld: in the Apple area of the Computer Center.

From Guy Kawasaki

Life is full of bizarre ironies. The one that bugs me the most—on an almost hourly basis—is the issue of the relationship between Apple and its software and hardware developers. Apple needs the help of developers to prevent Microsoft's Domination of Society (MS-DOS), and thousands of developers are willing to help in this battle. Yet, despite this apparent match of need and capability, there's this idea that the two often don't work well together, to the frustration of everyone.

My comments specifically address ways for you to get the most out of working with Apple; these principles hold true for anyone who wants to help—or get help from—the six-colored mothership. Here, in the most honest and up-front manner possible for a person who doesn't want to be an ex-Apple Fellow, are the rules for how developers can get the most out of Apple.

1. Don't take no for an answer. If there's a single misconception that drives developers crazy, it is that Apple is a single-headed beast. This isn't so. Apple has 12,000 employees, and every one of them thinks he or she is a vice president (except for the 12 who think they're president). Apple is like the Internet: You can't bring the whole system down, because each part works independently.

Believe it or not, the multiheaded nature of Apple is good, because if one part of Apple turns you down for anything, you can keep asking until you find someone who agrees. Just because one part of Apple says "no," it doesn't mean that a company-wide message has gone out or that a company-wide decision has been made. So keep asking until you hear what you want to hear.

2. *Match your product to the current Microsoft marketing thrust.* While Microsoft's system software is (at least) two years behind Apple's, its hype is two years ahead. If Microsoft announces a feature for Windows, you can bet that someone at Apple is worried that Apple doesn't have a response. Ironically, Apple probably had the idea two years before Microsoft but didn't capitalize on it. (First Apple copied Xerox's technology; now it's copying its approach to new technology—for sure I'll be an ex-Apple Fellow!) The bottom line: Watch what Microsoft announces, then tell Apple that you have the Macintosh version of the same idea, and the waters will part.

3. *Don't be proud, just get in.* There are developers who only believe in home runs. They want the worldwide marketing campaign to focus on their product just like PageMaker in 1985. They want to be bundled with every Macintosh. Anything less, and they are disappointed, frustrated, and angry. They can't understand how Apple can be so stupid as to not see things their way.

In the history of Apple, there's been only one PageMaker in the sense of company-wide focus to make a product a success. So you should take what you can get just to get inside the Apple kimono. The strength of Apple computer is in its soldiers: the thousands of employees who can help you in little ways. If you make enough "vice presidents" happy, one day you may find Vice Presidents making you happy.

4. *Understand the Apple employee's reality.* True or false: Apple employees all take two hours for lunch, drive BMWs, sit around in bean-bag chairs all day, and have no ideas about how to make Apple successful. False. Apple employees' lives are full of turmoil, budgetary constraints, and tsunami-like inundation of ideas coming from every direction—from their subordinates to their grandparents to their dentists. If you got to know them, you'd probably like and respect them—even the ones who set the prices for the developer services programs.

The touchiest situation occurs when developers come to Apple with ideas for penetrating markets. Apple gets more good ideas than it can comprehend, much less implement. For a close corollary, think of what it would be like to implement every feature request that every customer made for your software. (Please don't try to tell me that you do implement every request, because I've run a software company.)

5. Do what's right for your company. Even in my most evangelistic moments, I don't recommend that you do what's right for Apple, instead of doing what's right for your company. If you do things for Apple and your company fails, then you might say to Apple, "We did this for you, so now you owe us." Apple will deny it owes you anything, and the relationship will go into the toilet. Enlightened self-interest is the best path—then, if you're successful, you're successful. And if you fail, there's no one else to blame.

There you have it: the only guide to working with Apple you'll ever need. May you create the next PageMaker and be the next Aldus. If you do, and this column helped you, I might claim some credit for your success. ♣

Strategy Mosaic

Copland

continued from page 1

those opportunities, and why Copland will make it even more compelling to undertake international development efforts.

For those of you just joining the Macintosh developer community, Copland is the next major release of the Mac OS. Intended to provide a stable, modern platform for next-generation Mac OS computers, Copland is built around a microkernel and is being written almost entirely in "native" PowerPC code. Significant portions of Mac OS code are being rewritten so that Copland can provide greater stability and performance and preemptive multitasking services. For a complete overview of Copland, you'll want to read "Copland: Technology for Customers' Sakes" in the June 1995 issue of *Apple Directions,* available on the World Wide Web at http://dev.apple.info.com/appledirections/june.html.

Macintosh on the Move Outside the U.S.

It's an important part of Apple's strategy to continue to make the Mac OS the preeminent platform for international software development—and to encourage you to develop world-ready software, which can be released in more than a single geographic market at the same time. Why is this important? Consider the following facts:

- The Macintosh market outside the United States is increasing. Apple received 47.6 percent of its fiscal year 1995 revenues from outside the United States, up from 45.7 percent the year before.
- Japan is now the second largest Macintosh market, after the United States, and the Japanese market is growing. In 1994 personal computer sales in Japan grew 35 percent, and sales are expected to grow another 50 percent by the end of 1995, topping the 5 million mark, according to International Data Corporation. The Macintosh share of the Japanese computer market is growing:

According to the Software Publishers Association (SPA), Macintosh software made up 18 percent of total software sales in Japan in the second quarter of 1995, compared to 13 percent in Q2 1994.

• Asian software sales in the second quarter of 1995 generated more than 90 percent more revenue than sales in Q2 1994, again according to the SPA. Software sales are growing at well over a 100 percent annual pace in some markets, including India and Pakistan, New Zealand, Thailand, and China. These markets are small—software sales in India and Pakistan totaled just $4.8 million in the first half of 1995, while total sales for the same period were only $2.7 million in China—but they're experiencing phenomenal growth.

• Computer sales are expanding throughout the world into the consumer market, in which local users can't be expected to know more than their native language. To reach local consumers in other geographic markets, it's increasingly important to have products that are carefully localized for those markets. (In an extreme example of this, French commerce law now requires software products to be localized if they're to be sold in France.) Even if hardware and software sales outside the United States weren't expanding, Apple thinks it's good business to develop international, world-ready software. Apple has localized the Mac OS for more than 30 geographic markets, which span virtually every country where computers are sold in significant numbers. Your world-ready software can be localized easily to work in any of those markets.

For the incremental extra resources you'll spend making software world-ready, you get a product capable of earning revenues in many geographic markets at once. For many of you, this will mean developing an English-language version of your product, but developing it with the rest of the world in mind by using Mac OS WorldScript technologies. Doing so is not much more difficult than developing your product for a single market. On the other hand, if you develop with only a single different geography in mind—even if it's the burgeoning Japanese market—you end up with software that has to be significantly reengineered, at great expense, before it can be sold in other countries.

For those of you who develop first for other languages, what I said in the last paragraph holds doubly true, because your primary geographic markets will be smaller: You simply open yourself to more revenue opportunities by developing world-ready software, instead of developing for only one or two markets.

Expanding the World-Ready Lead

Copland technologies are intended to extend Apple's lead in letting you develop world-ready products. Copland will support world-ready software developed with any of the following three technologies:

• The WorldScript technologies, including the Script Manager and the Text Services Manager that work with any release of the Mac OS since System 7.1. Copland will support WorldScript to preserve today's investment in world-ready Mac OS software.

• The Copland Unicode interfaces. These as-yet-unreleased interfaces will let you implement some Toolbox functions using Unicode, the emerging standard that assigns every character in every modern writing system a single 16-bit code, thereby making the same program code able to manipulate text in any language.

• Text Objects, the Copland technology for managing text. Text Objects is a high-level application programming interface (API) built on both WorldScript and Unicode. This API borrows ideas from object-oriented programming that both make it easier to write world-ready applications and provide powerful new annotation features.

Additionally, Apple will soon ship a character-encoding converter, which translates non-Unicode character codes to Unicode, and vice versa. If they incorporate the converter, applications built to handle one type of character encoding can then work with data created with another character-encoding scheme. A beta version of the converter is expected to be included on the Developer CD by late Spring 1996.

These technologies will make it easier for you to develop more robust, higher performance world-ready software while also preserving your current investment. Current world-ready software, developed using WorldScript, will work with Copland—better than it works with today's Mac OS. That's because the WorldScript extensions will become an actual part of system software, and the parts of the system that provide world-ready features will be rewritten in native PowerPC code. World-ready software developed with Text Objects and/or the Unicode interfaces will run only under Copland, but it will enable features that today would require you to do a great deal more programming.

Today's Macintosh International Advantage

More about the Copland technologies in a moment, but first a little quiz: Which mainstream personal computer operating system lets you develop one application and release it simultaneously for all the world's languages?

Answer (pick one):
(a) Windows 3.x
(b) Mac OS
(c) Windows 95
(d) None of the above

If you picked "Mac OS," you were almost right. The correct answer is "None of the above"; in all honesty, it simply isn't possible now—and probably won't be in the forseeable future—to develop one version of an application and have it work with every language system, simultaneously, without at least some modification. Anyone who tells you otherwise is probably trying to sell you something you don't want to buy. Today's Mac OS, though, gives you the distinct advantage of letting you develop one globalized version of an application that can then be easily localized for all your target markets. This advantage is provided by the many localized versions of the Mac OS and the WorldScript technologies available since the release of System 7.1, including the Script Manager and the Text Services Manager.

Other parts of the Mac OS international advantage are Apple's Japanese, Chinese, Korean, Hebrew, Arabic, and Cyrillic language kits. These are add-ons to the Mac OS that let customers use the specific language supported by each kit on top of their system's primary language without having to install an entirely new system. Japanese language kits in the United States, for example, let customers work with Japanese software even though they're using the U.S. version of the Mac OS. See the box "How Apple's Language Kits Benefit Developers" (page 8) for more about the language kits.

The rest of the personal computer world is way behind Apple's international solution, and it's catching up only slowly. In fact, one could argue that the transition to Unicode is one way the rest of the industry is trying to pull even with the Mac OS. (For more about Unicode, see the box on page 7.)

With Windows 95, for example, you have to maintain three separate code bases if you want to be able to localize your application for all the world's major markets. You have to have one code base for simple 1-byte languages (for example, those that use the Roman writing systems, like English and French), a second for 2-byte languages (like Japanese and Chinese, whose complex writing systems require a 2-byte code for each unit of text), and a third for right-to-left writing systems (like Arabic). This can make maintenance of future software versions a hassle, to say the least, since you have to revise each code base and then relocalize each of them for your target markets.

With the Mac OS, though, if you develop your application according to the WorldScript programming conventions—making it world-ready—a single code base can be used for localization in all the world's markets. For the relatively small incremental investment of adopting WorldScript, you can localize and release your product simultaneously in more than a single market, and get additional revenues.

Say your primary market is the United States; if you don't adopt Mac OS international technologies, you have to reengineer your product—perhaps significantly—if it's to work in Japan, or Israel, or Russia, or another country that uses a different writing system. (Of course, that product is no worse off than an average Windows program, which generally requires heavy recoding if it's to be released for more than a single language system.) However, if you adopt WorldScript, you can then localize your product fairly easily for release in those other countries, and the investment you've made in developing your product can result in far greater return.

Globalization and Localization

Note the subtle, but important distinction between *globalized,* world-ready software and *localized* software. Globalized software has been prepared with the potential needs of the world's users in mind. It's been prepared to be "culturally neutral"—that is, it uses interface elements that can be easily understood no matter where in the world they are viewed and text strings and menu items that work well no matter what language they are translated into.

A globalized application can be more easily localized or customized for a special market since it is capable of using any script system and already works, technically speaking, in the language for which it is being localized. To quote *Inside Macintosh:Text,* "Globalization involves careful design and writing of the application and its textual and graphic resources."

Localized software has actually been prepared for release in a specific market. All text in the software has been translated to the writing system of the target market, as has the documentation. Localization also includes additional changes that are necessary to make your software culturally acceptable to the target market.

The WorldScript Opportunity

Ease of global development remains one of the clearest Macintosh advantages, and it sounds as if the adoption of WorldScript would be an obvious choice, right? Surprisingly enough, although many developers take partial advantage of WorldScript to ease localization for the large Japanese market, relatively few developers, large or small, take full advantage of WorldScript to make their products world-ready. This means that there remains a huge opportunity for you if you want to adopt a truly global Macintosh software development and publishing strategy. If you do so, your programming investment will be much more highly leveraged than your competition's—and you'll have a shot at profits and success that your competition doesn't have.

Under Copland, your global development efforts can become more robust. You'll be able to implement more features than under the current Mac OS architecture, and you'll be able to implement many features more easily than today. Before taking a look at Copland, let's spend a minute or two talking about the existing Mac OS globalization technologies, because those technologies will remain a part of the Mac OS when Copland is introduced.

The WorldScript Programming Discipline

Apple uses the term *WorldScript* to encompass the technologies for developing world-ready software that became part of the Mac OS with the release of its own world-ready version, System 7.1. *WorldScript* actually defines a general programming discipline or a methodology rather than a specific API. That's one reason it hasn't been universally adopted; it's not a specific set of calls and routines documented in a volume of *Inside Macintosh* that you can follow like a recipe. Instead, it's an approach to programming and software design—including human interface design strategies, specific Mac OS APIs, and just plain common sense. If you learn and follow that approach, you can develop software that's much easier to localize for specific languages than software you've developed with a single language in mind. See "World-Ready Basics" on page 9 for a list of resources your programmers and designers can use to learn the WorldScript discipline.

In discussing WorldScript, it's important to distinguish WorldScript itself from the WorldScript I and II extensions that control world-ready software features. The WorldScript extensions are important parts of the WorldScript programming method.

By itself, the Mac OS controls the display, manipulation, output, and so on of the Roman writing system. The WorldScript I extension builds in similar controls for so-called 1-byte complex script systems. These are systems, like Hebrew and Arabic, whose characters can be represented by a single byte of code, but employ a complexity beyond that of the Roman writing system, such as contextual characters or right-to-left line direction. WorldScript II contains the same functions for so-called 2-byte writing systems—like Chinese and Japanese—whose complex characters (or "glyphs") need to be represented by 2 bytes. Apple developed WorldScript I and II so your software could call on their services instead of your having to implement similar services separately in each application.

Script Manager and Text Services Manager

The way your application calls on WorldScript I and II services is through the various Mac OS text-related managers, the most important of which for this discussion are the Script Manager and the Text Services Manager. (All the text-related managers are fully described in *Inside Macintosh: Text*). The Script Manager is the standard application interface to script systems, providing a set of functions that let an application get information about its text characters. If you haven’t adopted it yet, we strongly urge you to do so next time you revise your application. Here’s why.

Hard-wiring an application for a single writing system (Roman, Arabic, Japanese, and so on) or, instead, building in calls to the Script Manager are comparable amounts of work. Hard wiring may be simpler in the short run, because it doesn’t require you to consider how a variety of languages might handle text; you only need to think about the language you know best. However, if your application calls the Script Manager, the application can then determine what it needs to know about the writing system being used, including date and time format and information for text display and output (that is, printing). Incorporating the Script Manager makes your application much more flexible, because it can then handle any modern writing system.

The Transition to Unicode

Unicode has become an official industry buzzword. We all know that in the near future, any personal computer of merit will use Unicode. But it seems that only those people with a solid understanding of how computers work with text really know what Unicode is, and why the industry is moving in that direction. The following is a brief attempt to help you understand the motivation behind the move to Unicode. It won’t make you an expert, but it will help you the next time the subject comes up at a cocktail party or a meeting.

Unicode is the industry’s effort to arrive at a universal character set that encompasses every letter and symbol from every imaginable language. Supported by Apple, Microsoft, IBM, Digital, Hewlett-Packard, Lotus, Novell, and others, it’s the first international multivendor standard for character sets. Its repertoire includes all modern writing systems (and some archaic ones, as well). Within Unicode, each character from each writing system is assigned a single, unique 16-bit code, allowing Unicode to accommodate more than 65,000 different characters. (The Unicode consortium has determined methods for Unicode to accommodate even more characters than that). This means that any Unicode-compliant system will always recognize the code for each character.

This is a significant change from today’s situation, in which just about every computer platform uses a different set of codes to represent the letters and symbols of different writing systems. The Macintosh, for example, uses 1-byte codes for writing systems that employ letters (like Roman and Cyrillic) and 2-byte codes for character-based systems (like Chinese). This works for exchanging data between similar computers with operating systems employing the same language systems. If different types of computers employing different languages exchange data, however, usually one computer can’t interpret the information from the other computer.

For example, today a Mac OS computer running the U.S. version of System 7 will take its best guess at interpreting text data from a Japanese system but be unable to read it, unless it uses special translation software. That’s because the code used to create text for the U.S. Mac OS system doesn’t mean the same thing to the Japanese computer. For an example of what happens when a U.S. system tries to read Japanese text, visit Bandai’s Pippin home page on the World Wide Web, which was created using a Japanese character system. (Go to http://www.fix.co.jp/bandai/pippin.html and click Software Information.) If you’re running a non-Japanese version of the Mac OS (and you don’t have the Japanese Language Kit installed), the text looks like this:

```
    $B:#$^$G$N%\!<%I%2!<%`$N>o<1$r$/$D
$,$($9!”(J $BD6H~No$JI!*(J$B:#2s$OA*$Y
$k#4?M$N%&%k%H%i%R!<%m!<$H#3#0BN
0J>e$N?M5$2x=C$,EP>I$7$^$9!*(J$B:#$^$GL
55!e$;$^$7$H=D2#L5?T$KF0$-$^$o$k!*(J$B
M=Dj2A3J!o(J5800($B@GJL!K(JCopyright
BANDAI Co.,Ltd ,$B1_C+%W%m(J 1995
```

Tomorrow, once the entire industry makes the transition to Unicode, the Bandai site will use the Unicode character set, the Mac OS will be able to read it, and under the same circumstances, you’ll see Japanese characters instead of gobbledygook. Any text data will be transferable from one type of computer to another, and, regardless of the writing system native to either computer, the data will still have the same meaning. Every character from every language will have its own unique code, and every computer will know exactly what that code means.

The implementation of Unicode will require a fair amount of work from many people and companies. The data types for text handling under Unicode are quite different than in today’s mainstream environments, meaning that major portions of both operating systems and applications will have to be redone to incorporate Unicode. According to Apple international evangelist John McConnell, translating an application to Unicode text strings will be an amount of work on the order of redoing a 680x0 application so it can run in native RISC mode with Power Macintosh systems.

Once that work is done, Unicode will help immensely in international, cross-platform software development. The same code base will at least be readable by any computer in any language. However, it’s not a panacea, because even Unicode-based software won’t be immediately useful in any language. It will still have to be localized for different markets.

In many ways, Unicode will give other platforms what’s already available on the Macintosh today. Using WorldScript when you develop Mac OS applications makes them world-ready and easy to localize; in the future, using Unicode when developing any application will make it much more world-ready and easily localizable for any platform (Mac OS, Windows, OS/2, UNIX®, and so on) and any language.

Unicode will also help the current Mac OS international technologies. Currently, the Mac OS supports multiple character-set encodings in different countries. The single, universal Unicode encoding will replace all of these. Unicode will make it possible for the Mac OS to do without the WorldScript extensions, which currently keep track of the encodings for the world’s complex writing systems. Also, you won’t need language kits to read character sets other than the one that your system uses. Apple is committed to supporting Unicode, and it will begin the transition of the Mac OS to Unicode under Copland, completing the transition with Gershwin, the version of Macintosh system software coming after Copland.

If you’d like to learn more about Unicode, we recommend a book prepared by the Unicode consortium called *Unicode Standard Version 2.0,* which was just about to be published by Addison-Wesley as we went to press. (The version 1.0 volume is currently available.) You can also find information at the Unicode home page on the World Wide Web (http://www.stonehand.com/unicode.html).

You also need to adopt the Text Services Manager (TSM) in your application if you want it to be world-ready. The TSM manages the communication between applications and utility programs that provide text services. Currently, the most frequently used text services are input methods, which convert phonetic or syllabic characters into ideographic or other complex representations. Input methods let you use a standard keyboard to generate the thousands of characters used by some languages.

Applications using Japanese, Chinese, Korean, and other 2-byte writing systems require the use of an input method. There are two basic types of input methods for 2-byte systems: floating-window input and inline input. Floating-window input methods require you to create text strings in a special "floating window" and then transfer the text to the place you'd like it to appear, whether it's in a dialog box or the application window. Inline methods let you type text strings directly where you'd like them to go. Before System 7.1 and the Text Services Manager, the Mac OS supported only floating-window methods; now, with the TSM, Mac OS customers can use the far more convenient inline input methods. If your application is going to let your customers use inline input—and if it's going to be localizable for Japan, the second largest Macintosh market—it will need to incorporate the Text Services Manager.

There are two levels of TSM support; the first level is called *TSM-aware.* With very little

How Apple's Language Kits Benefit Developers

Apple has just added three new language kits to the language kit product family, which means markets for some of your localized applications have just expanded globally. The language kit product family, which was introduced in 1993, further extends the Macintosh computer's multilingual, international advantage. The Hebrew, Cyrillic, and Arabic Language Kits have joined the Japanese and Chinese Language Kits in December, with Korean soon to follow.

The kits let users enhance their systems with additional language capabilities that let them input, edit, and print text in additional writing systems (or languages), using world-ready software, whether it's a localized application, or a multilingual application that has support for 2-byte characters, right-to-left text, or both. Also, customers can add one or more writing systems of their choice to the "native" writing system in use with their version of the Mac OS.

The kits are intended for use mainly by specific groups of customers in the following areas:

- in education—by students learning a new language, and by their teachers and language labs
- in expatriate communities—for example, Japanese natives living in the United States
- in small businesses that do multilingual publishing, prepare subtitles, or undertake translation
- in the software development community—by developers who need assistance in their international development efforts

The language kits expand the business opportunity for localized applications. Since the language kits ship worldwide, the market for, say, a Korean application will no longer be limited to Macintosh customers in Korea. A Korean application can run either on the Korean system that ships with Macintosh computers in Korea or on any Macintosh, anywhere in the world, that is running the Korean Language Kit.

Additionally, the kits provide an efficient way for you to test globalized and localized software. You can run the language kits on top of your Macintosh system—as long as it's System 7.1 or later—to find out if your product will work well with a given language. That means you don't have to install the version of the Mac OS for that language over your system or maintain separate hardware dedicated to specific languages. (Final testing is recommended in the fully localized environment.)

The language kits' main contribution is that they allow the Macintosh to handle more than one language without sacrificing any other capabilities. For example, users won't have to choose between running an English or a Japanese version of the Mac OS; with a language kit, they can have both on a single system.

Each kit includes fonts, input methods, and language resources for the particular language used by the Script Manager, which tells the Macintosh how to handle more than one script. In addition, the kits contain a language register that allows the Mac OS to open localized applications using the correct language (actually the fonts for that language) in menus, dialog boxes, and other elements of the user interface, regardless of whether or not the region code in the 'vers' resource has been correctly set. When you localize an application, you're supposed to set the region code to indicate the language you're localizing for; the region code tells the system to open the application's user interface in that local language.

The demand for the language kits is driven by the fact that all residents of a single geography do not speak only one language (or use only one writing system). This has long been true in Europe and is becoming increasingly so in the United States. According to the U.S. Census Bureau, 1 out of every 14 people in the United States speaks a language other than English at home.

People who speak languages that use non-Roman writing systems, but live in a region where another writing system is used, are the main target for language kits. The largest such market—more than 5 million, according to Apple's latest estimates—is made of Chinese speakers living outside China. This includes roughly 1 million Chinese speakers living in the United States, 500,000 in Japan, 2 million in the rest of Asia, 500,000 in Canada, 250,000 in Australia, and 1 million in Europe. All the other writing systems supported by language kits—now including modern and traditional Chinese, Japanese, Hebrew, Yiddish, Arabic, Persian, Cyrillic, and soon Korean character sets—are used by significant numbers of people outside their native region.

To work with the language kits, your software must be prepared according to the WorldScript discipline (or QuickDraw GX line layout/typography methods). Your product doesn't need to be localized to work with the language kits; it just needs to be WorldScript-savvy. The kits greatly increase the market f or your world-ready applications, giving you another solid business reason to adopt WorldScript.

[Editor's note: This sidebar is based on an article that appeared in the August 1993 issue of Apple Directions.*]*

knowledge of Japanese or other 2-byte systems and a few dozen lines of code, you can make your application TSM-aware. This means that inline input methods will work properly with the elements of your application controlled by TextEdit, such as dialog boxes. If you don't make your application TSM-aware, Japanese (2-byte) users won't be able to use the latest inline input methods when they save a file, for example; instead, they might have to use Roman (1-byte) filenames.

If your application requires heavy use of text—for example, if it's a word processing or desktop publishing product—it will have to be TSM-savvy, which is the second level of TSM support. TSM-savvy applications support the use of inline input methods in the application window itself. The amount of work it takes to make an application TSM-savvy depends on the application itself; applications that support wide varieties of text formatting, styles, and so on, can require a great deal of work.

If you go to the trouble to make your application TSM-savvy, it can handle text from any TSM input method, for any complex writing system; you only have to do the text services part of your programming once. If, on the other hand, you hard-code your application to handle only a specific input method for the Japanese character set, and you later want to release that application for the Chinese and Hebrew markets, for example, you then have to recode your entire application twice, once for each market.

Other Aspects of WorldScript Programming

Adopting the Script Manager and the Text Services Manager are the technical cornerstones to making your application world-ready. There are other aspects to the WorldScript programming discipline that you'll also need to follow if you're to make your software world-ready. As I said earlier, WorldScript is as much an overall approach to application design as it is a technical solution. It includes the kinds of programming techniques you'll learn in *Inside Macintosh: Text* about the Script Manager, the TSM, and several other text-management technologies.

WorldScript also includes an overall approach to designing your application, one that keeps in mind the needs of the various cultures that one day might be in the market for your product. To begin to familiarize yourself with this kind of cultural sensitivity, you'll want to read Apple's *Guide to Software Localization* and the sections in the *Macintosh Human Interface Guidelines* that pertain to international software, especially Chapter 2, "General Design Considerations." The Macintosh Technical Note "OV 20—

World-Ready Basics

If you want to make your application world-ready and easily localizable for the world's major personal computer markets, here are the general steps to follow. This is a gross oversimplification, intended to get you started if you're not already familiar with WorldScript and software globalization. However, if you study the relevant documentation closely, you'll be well on your way to understanding what you need to do to develop world-ready software. (Unless otherwise noted, all the documents mentioned here are available from APDA; see page 36 for APDA ordering information.)

1. Adopt the Script Manager. This will mean that your application can handle virtually any modern writing system (for example, Roman, Arabic, Cyrillic, and Japanese) through the WorldScript I and II extensions. The Script Manager is documented in *Inside Macintosh: Text.*

2. Adopt the Text Services Manager (TSM). At least make your application TSM-aware; this means that any element of your application that uses TextEdit (such as a dialog box) can use the services of TSM input methods for complex writing systems. If your application also makes heavy use of text, make it TSM-savvy so that text from input methods appears properly within the application window itself. *Inside Macintosh: Text* contains what you need to know about the TSM.

3. Make appropriate use of miscellaneous text-management resources. Some of these are documented in *Inside Macintosh: Text;* others in the technical note "OV 20—Internationalization Checklist."

4. Use QuickDraw GX instead of QuickDraw for text display. If you do, your application will work with almost any character set.

5. Read the *Macintosh Human Interface Guidelines,* especially Chapter 2, "General Design Considerations," and the *Guide to Software Localization,* to be sure your application is ready for localization in any language. You can find them both on the Internet at ftp.info.apple.com/Apple.Support.Area/Developer_Services/Technical_Documentation.

6. Obtain the preliminary documentation for Text Objects, the Copland API that will become Apple's technology of choice for developing world-ready software, and study it to decide what Text Objects can do for your application. You can find it on the 1995 Worldwide Developers Conference Technology CD; if you don't have that, we've posted it for you at the *Apple Directions* Web site (http://dev.info.apple.com/appledirections/adextras/adextras.html).

7. Watch *Apple Directions* for more news about the release of the Unicode interfaces for Copland.

8. Take a look at the beta version of the character-encoding converter for converting between Unicode and non-Unicode strings to determine whether you want to incorporate it in your application. The converter, which will be released on the Developer CD Series within the next few months, will be a standard part of Copland.

9. Read up on Unicode so that you can understand what you'll need to do to make the transition; see the resources listed at the end of "The Transition to Unicode" (page 7).

10. Consider hiring someone with knowledge of one of the character-set types that your company lacks expertise in (simple 1-byte, like Roman; complex 1-byte, like Arabic; 2-byte, like Japanese, Chinese, and Korean). This will help you keep world-ready software design in mind whenever you plan new releases, and help assure that your software can at least run in one of the other environments.

11. If you don't have the resources to do your own localization in your target markets, find a localizer for that market. If you develop for Europe, see AppleLink for a list of localization companies [path: Europe:Selling Into Europe]. The following Apple employees can also recommend localization companies for your markets:

• for Japan, Takeuchi-san (Takeuchi2 on AppleLink, or takeuchi2@applelink.apple.com)

• for the rest of Asia, Moses Dharma (DHARMA.MOSES on AppleLink, or dharma .moses@applelink.apple.com)

Internationalization Checklist" also contains technical and nontechnical details you'll need to think about when you develop world-ready software. (See "World-Ready Basics" for information on these and other documents.)

When it comes to developing world-ready software, there's also no substitute for having the right people on hand to help. It's a real plus if you can have someone on your staff with expertise in at least one of the writing systems you hope to target. Say that you're most familiar with English and the Roman writing system. It's a good idea to have someone on staff who, at a minimum, has expertise in *either* 2-byte systems (like Korean, Japanese, and Chinese) or the needs of complex 1-byte systems (Arabic, Hebrew, and so on). This person can review your early design decisions to be sure that, when you try to localize your product for its new markets and writing systems, it will at least work well with the one in which they have expertise.

It's probably a luxury for most of you to employ the number of people needed to cover the world's major writing systems. By hiring someone who is expert in at least one of them, you'll have a leg up on the globalization/localization process, because you'll be sure to consider world-ready design issues when you develop your products. For languages your staff doesn't cover, you can always find consultants and localization companies to help. (See "World-Ready Basics" for information on finding that sort of help.)

Another technology you need to pay attention to in developing world-ready software is QuickDraw GX. If you haven't yet adopted QuickDraw GX, the demands of globalization give you one more reason to do so. That's because QuickDraw GX (unlike the earlier QuickDraw) works with any character set, and has already been globalized. In fact, QuickDraw GX supports Unicode. If you display text with QuickDraw GX instead of with QuickDraw, you almost get support for complex 1-byte and 2-byte languages for free. If you use the earlier version of QuickDraw, your application will need to make many more calls to the Script Manager to display text from other writing systems.

What Copland Adds to the Picture

Under Copland, all the technologies just described will work, meaning you can continue to capitalize on the world-ready software you've already developed. That's the good news. The better news is that current WorldScript software will work even better under Copland, for two main reasons: First, instead of being extensions to the system, WorldScript I and II will be part of the core system, so there won't be any system patching under Copland to provide world-ready features. Second, more than 90 percent of system software, including all parts that control international features, will be rewritten in native PowerPC code. (Note that the WorldScript I and II extensions have been fully native since the release of System 7.1.2.) These two changes will result in greater stability and performance.

Additionally, as I mentioned already, Apple will release several new technologies between now and the release of Copland that will be part of the Copland version of the Mac OS. One of the main reasons behind the new technologies will be to support the industry's transition to Unicode. Similar to the way Copland will begin the transition to preemptive multitasking by enabling some functions to be run as preemptive tasks in their own protected memory space, Copland will begin to introduce Unicode to the Mac OS. It's expected that the transition to both Unicode and preemptive multitasking will be completed with the next version of the Mac OS—Gershwin.

One Mac OS technology that will help ease the transition to Unicode is the character-encoding converter. It will be released separately before Copland but will become part of the new Mac OS. The converter contains two modules, one that converts Unicode into non-Unicode and one that works the opposite way. If you incorporate the first module into your Mac OS application, it will be able to convert Unicode strings into the character encoding used in the application so that your application can "understand" Unicode. Similarly, if you incorporate the converter into a Unicode application—say, one that's created under Windows NT that you're porting to the Mac OS—it will "understand" and be compatible with the Mac OS character codes.

Using the Copland Unicode interfaces, which won't be available until later in 1996, you'll be able to write Unicode strings to call some—but not all—Toolbox functions. The interfaces will let you work directly with Unicode strings, so you can actually program parts of your application using the Unicode character set. Coding with Unicode, however, can be slow going and tough work, so many of you won't be interested in the interfaces.

There is a Copland technology for letting the Mac OS work with Unicode strings that, in typical Macintosh fashion, will make life much easier for you. That's Text Objects, the new Copland text-management API that will allow the Toolbox to handle text from different encodings, including Unicode. Text Objects is built on top of both Unicode and WorldScript, giving you a high-level way of managing text, and the entire Copland Toolbox will have Text Objects interfaces.

As I've already said, Copland will support existing WorldScript technologies to be sure your existing investment in development is preserved. Apple expects, however, that the more robust, object-oriented Text Objects method of supporting multilingual text, using multiple encodings, will eventually make the existing methods for software globalization obsolete. Text Objects gives you a much different way of developing world-ready software, one that will take some learning on the part of your programmers. However, before using the older WorldScript technologies or the Unicode interfaces, you'll want to ask yourself, "Can Text Objects do what I want?" The answer will often be "yes," because it does a lot.

In addition to identifying the encoding of a text string, Text Objects will let you bundle other information with the raw text; although there's no limit to the types of information that can be bundled, Apple envisions Text Objects to be used for data related to the text itself, such as the language it's in, pronunciation data, the actual sound made when someone speaks the text, or information that can be used to sort the text.

For example, Text Objects can come in handy if you want your application to be able to sort files in languages without alphabets. Today, if you open a folder full of files with names that were created with the Japanese character set, there's no convention for sorting them by name. Since Japanese doesn't employ an alphabet, there's no such thing as alphabetical order. However, if the phonetic pronunciation of each name is attached to the text, you can sort them according to pronunciation, enabling Japanese users to provide more order on their desktops.

Here are a few other examples of how Text Objects can be used:

• In handwriting recognition systems, you can store the "ink" version—what the user actually writes on the screen—with the cleaned-up text.

• You can create a text string with information for two writing systems, so the same string can be opened in two languages.

• You can store the information contained in a file in the filename itself.

Apple shipped preliminary documentation about Text Objects on the 1995 Worldwide Developers Conference Technology CD, and will continue to provide information about Text Objects long before the actual release of Copland. In other words, you can start getting ready for Text Objects, and figure out how you can use it today, and have world-ready Text Objects software ready to go by the Copland ship date. (The preliminary Text Objects documentation is also available on the World Wide Web; see "World-Ready Basics" for its location.)

Text Services Manager Under Copland

One part of the Copland international story will require a very few of you to undo and redo software you've created under the System 7 architecture. That's the new Text Services Manager. The few of you who've already developed TSM services—primarily input methods—will have to rewrite them for Copland because of the new process model it will introduce. However, Apple thinks it will be more than worth it because of the improvements being made to the manager. The rest of you will have still more reasons to adopt the Text Services Manager in your products. (If you're among the elite group of input method developers, Apple will be presenting briefings to help you redo your software so it works with Copland. For information on how to take part in a briefing, contact Apple's international evangelist, John McConnell, at J.MCCONNELL if you use AppleLink, or j.mcconnell@applelink.apple.com if you use another e-mail service.)

Under the current Mac OS, the TSM is a text services manager in name only; it's really mostly an input method manager. Broadly defined, a true text service replaces one chunk of text with another chunk of text. So, text services could include spell checking, in which improperly spelled text is replaced by properly spelled text; grammar checking, in which grammatically poor text is replaced by grammatically sound text; or any type of translation. The Copland TSM will support all those services, and any other type of text service you might imagine.

One interesting application of the new TSM might be a service that replaces shorthand with the text it represents. A legal secretary might record a deposition using shorthand on a Newton device, and then use a specially created TSM service to "translate" the shorthand symbols into the text they stand for, saving the time it might normally take to type the deposition.

Make Your Software Ready for the World

That, in a rather large nutshell, is what you'll need to know to globalize your software, and to take advantage of the increased revenue opportunities available to world-ready products, now and after the release of Copland. Of course, you must still localize your software, or have one of many available localization companies do the work for you. (See "World-Ready Basics" to find out how you can get in touch with localization companies in your target markets.) If you go to the trouble to learn the WorldScript programming discipline outlined here, however, localization will be far easier and less expensive than if you develop with only one geographical region in mind.

Learning WorldScript and the Mac OS international technologies—today's and tomorrow's—may take some doing, but it's less work than the reengineering it takes to move an application that's hard-coded for one writing system into a new writing system. The incremental costs of adopting WorldScript are leveraged into a product that can be simultaneously localized and sold in every Macintosh market around the world. If you haven't made your software world-ready yet, Copland technologies will give you even more reason to do so. Remember that adopting WorldScript and the Copland international technologies will give you the world—literally on a text string. ♣

Apple News

COMDEX

continued from page 1

previously known as the Common Hardware Reference Platform (or CHRP), which Apple released jointly with IBM and Motorola

• the Newton 2.0 operating system—which won a Best of COMDEX award—and two new Newton partners

• OpenDoc for Mac OS 1.0 and the OpenDoc for Mac OS Software Development Kit (SDK), which contains the completed OpenDoc software as well as sample code and tools for development of OpenDoc-based solutions

• a prerelease version of Apple's innovative SD-ROM (super density–read only memory) drive; SD-ROM discs can store up to 15 times as much data as current CD-ROM discs

• prototype PC Compatibility Cards with Pentium and Cyrix 586 microprocessors running on a PCI-based Power Macintosh system

• the Apple Internet Server Solution, which now includes Adobe™ PageMill, Adobe's easy-to-use Web page authoring software

Details about these announcements and demonstrations follow.

PowerPC Platform Specification

Perhaps the most eagerly awaited announcement was the availability of the specification formerly known as CHRP, the common hardware reference platform built around the PowerPC microprocessor. Now called the *PowerPC platform,* the specification defines a unified 32-bit personal computer architecture that will make it easier for system vendors to design computers capable of running multiple operating systems, including the Mac OS, AIX, OS/2, Windows NT, NetWare, and Solaris.

The specification provides a variety of benefits for developers: Mac OS software vendors will be able to market their products to a broader market, including customers who buy PowerPC processor–compliant systems primarily to run non-Macintosh software. Additionally, the specification will enable a great many more software developers—on whichever OS they prefer—to take advantage of the superior performance provided by RISC PowerPC microprocessors, including the floating-point unit. Additionally, vendors of PowerPC processor–compliant hardware systems will be able to create a single product that can reach multiple OS markets.

The specification details input/output interfaces, bus standards, and other system-level functional elements that hardware vendors will need to employ to build PowerPC platform–compliant systems—ranging from portables to high-end servers—based on the RISC PowerPC processor. The PowerPC platform specification uses widely available, industry-standard hardware components, which should result in lower-cost systems that can be readily manufactured by a variety of vendors.

Apple, IBM, and Motorola have already announced plans to deliver PowerPC platform–compliant systems, along with Canon, DayStar Digital, FirePower Systems, IPC Technologies, Pioneer, PowerComputing, Umax, and Zenith. The first PowerPC platform systems are expected to ship in the second half of 1996.

You can obtain the specification from Apple's World Wide Web site (http://chrp.apple.com), or by calling Apple (800-251-8662), IBM (708-296-6767), or Motorola (512-434-1502).

Newton 2.0 And New Newton Partners

As we reported last month, Apple previewed the Newton 2.0 operating system at Comdex.The new version of the Newton OS includes significant enhancements, including improved handwriting recognition and communication features, more built-in applications, and more efficient, less-expensive development tools; for the complete story, see "Newton—Surviving and Thriving at Age 2" in the the December 1995 *Apple Directions*.

What we didn't know yet was that Newton 2.0 also garnered top honors at the show, receiving the Best of COMDEX award in the operating system category. The coveted Best of COMDEX awards, presented annually by *Byte* magazine, recognize unique, technically innovative and cutting-edge new computer products at the show.

Additionally, Apple announced two new Newton technology licensees in Las Vegas: Schlumberger Electronic Transactions and Digital Ocean. Schlumberger Electronic Transactions, a division of Schlumberger Ltd., plans to use Newton technology in integrated "smart card" systems, which are designed to automate medical transactions. Digital Ocean, a developer of wireless connectivity products for Macintosh and Newton systems, plans to incorporate Newton technology with Global Positioning System (GPS) and wireless communication features in specialized applications in the manufacturing, transportation, health care, and services industries.

Both licensees should further expand the market for Newton software products. The Newton MessagePad is currently the leader in the personal digital assistant market, holding a 58 percent share in 1994, according to BIS Strategic Decisions.

OpenDoc for MacOS Introduction: More Than 29,000 Served

Another major milestone announced at COMDEX was the availability of OpenDoc for Mac OS 1.0 and the OpenDoc for Mac OS SDK at the Apple Web site; again, full coverage of the announcement can be found in the December *Apple Directions* (see "OpenDoc 1.0 for Mac OS Goes Golden Master").

We'd say the release has been an overwhelming success: In the 19 days after the SDK was posted on the site (which happened November 10), 29,480 different users viewed information there, according to statistics tracked by the OpenDoc Web site server. Also, there were more than 48,000 requests to download documentation, software, and other parts of the OpenDoc product release from the site. In those 19 days, more than 55 gigabytes of data were exchanged between the server and Web users. In case you haven't downloaded your copy, you can find it on the Web at http://www.opendoc.apple.com.

Another highlight of the OpenDoc 1.0 release was Claris Corporation's announcement that it will join Component Integration Laboratories (CI Labs), the vendor-neutral industry consortium promoting OpenDoc development, and deliver its first OpenDoc-based products in 1996, joining the 300-plus developers who have already committed to shipping OpenDoc products. Said Guerrino De Luca, president of Claris Corporation, "Claris intends to ship one or more OpenDoc-compliant products in 1996 and is evaluating opportunities to add OpenDoc container and component capabilities across our product line."

SD-ROM: The Multimedia Disc of the Future

Apple also announced—and demonstrated—its commitment to the new super density (SD) disc standard, which was announced by the SD Alliance earlier in the fall. Apple representatives showed a Toshiba SD-ROM drive connected to a Power Macintosh 6220 playing a disc that combined the content of six CD-ROM titles.

The new SD-ROM discs can store about 8 to 15 times more than today's CD-ROM format. An important feature of SD-ROM drives is that they can play existing CD-ROM discs and music CDs, so you'll be able to continue to sell CD-based content and software that you've already developed once the new standard is in place. Apple intends to include the new devices with Macintosh systems as soon as commercial SD-ROM drives are available in sufficient quantity.

It's expected that SD will become a new distribution method for movies without the degradation of quality suffered with traditional video tape. SD will also result in more compelling multimedia titles, with faster, smoother playback of video, providing consumers with a more TV-like interactive experience. Market research firm InfoTech of Woodstock, Vermont, projects that worldwide unit sales of SD drives based on a 4.7 gigabyte format will exceed 2 million across all applications in 1997, the first full year they'll be available. InfoTech forecasts that 1.2 million high-density drives for personal computers will sell that year. InfoTech expects that in the year 2000, approximately 39 million SD drives will be sold, accounting for nearly one-third of all CD-ROM drive sales that year.

Prototype Pentium PC Compatibility Cards

At COMDEX, Apple also demonstrated a Power Macintosh system with the PCI (Peripheral Component Interconnect) bus, running prototype PC Compatibility Cards featuring Pentium and Cyrix 586 processors. The prototype PC Compatibility Cards showed that PCI-based Power Macintosh systems have the potential to run Macintosh, DOS, and Windows software, providing wider compatibility than any other personal computer system. Apple currently provides DOS and Windows compatibility through 486DX2/66-based DOS Compatibility Cards on selected models of its Power Macintosh, Macintosh LC, and Macintosh Performa lines of personal computers.

According to a 1995 Apple Early Buyer Study reported on in the August 1995 *Apple Directions*, Apple's most popular cross-platform system, the Power Macintosh 6100/66 DOS Compatible, has attracted a large number of customers replacing an x86-based PC. Additionally, DOS-compatible Macintosh systems offer Windows 95 users an ease-of-use advantage over Pentium-based and other Intel x86-based systems: They enable users to switch easily between Windows 95 and Windows 3.x. Most PCs won't let customers maintain both versions on the same system.

Adobe PageMill Bundled With Apple Internet Servers

Last but definitely not least, Apple announced that its Internet servers are becoming more powerful with the addition of PageMill, Adobe's new Web page authoring tool. The software is now being bundled with the Apple Internet Server Solution, which consists of a PowerPC processor–based Workgroup Server 6150/66, 8150/110, or 9150/120 and a CD-ROM. The package also includes WebSTAR Web server software from StarNine Technologies, Inc., and the Netscape Navigator Web Browser from Netscape Communications Corp. PageMill lets you create, edit, and update Web pages without experiencing the complexities of HTML (Hypertext Mark-up Language).

With PageMill, you can drag and drop text and images, apply styles, and import graphics and textures into a Web page as well as easily create links between Web pages. If you bought one of Apple's Internet Server Solutions after September 1, 1995, you can receive a copy of Adobe PageMill for only the cost of shipping and handling by calling 408-862-3385 (qualified customers only, please!).

Apple Ranks #1 in Personal Computer Sales in the U.S.

Apple received more good market share news at COMDEX: According to announcements made there by two respected research firms, Apple shipped the most personal computers in the United States in its fourth fiscal quarter of 1995, or Q4 '95 (that is, July through September).

Both Dataquest, Inc., and International Data Corporation (IDC) put Apple Computer, Inc., in first place in their quarterly surveys of computer sales in the United States for the period. Apple topped all competitors; Macintosh computers made up over 13 percent of all computers sold in the United States during the quarter. Dataquest pegged market share at 13.1 percent, while IDC estimated share at 13.9 percent.

According to the IDC data, Apple gained market share between Q4 '94 and Q4 '95. Macintosh computer sales grew faster than the overall industry between the two quarters. While overall U.S. computer sales increased 21 percent—from 4.7 million to 5.7 million—between Q4 '94 and Q4 '95, Macintosh unit sales increased 25 percent—from 637,000 to 795,000.

"Contrary to popular belief, Apple is not losing market share," said Eric Lewis, IDC's manager of Personal Systems Research and its principal Apple analyst. "They have maintained market share so far this year, and our projections are that the company will gain share in Q1 '96 [October through December]. Sales to the education market and introduction of new PowerPC processor–based systems for both homes and schools contributed to their strong showing in Q4 '95, which is up 25 percent over the same period a year ago."

"Apple's strategy of focusing on its key markets appears to be paying off," said Kimball Brown, Dataquest's vice president of Personal Computers. "Strong education sales and high demand among consumers pushed Apple into first place in the United States, and helped grow the company's market share worldwide. We expect a very strong showing in the current quarter and holiday quarter as well, as Apple enters the quarter with their guns loaded."

Top Five Q4 '95 U.S. Personal Computer Vendors

IDC data

		Units shipped (000s)	Market share (%)
1	Apple	795	13.9%
2	Packard Bell	705	12.4%
3	Compaq	670	11.7%
4	IBM	490	8.6%
5	Hewlett-Packard	305	5.3%

Dataquest data

		Units shipped (000s)	Market share (%)
1	Apple	788	13.1%
2	Compaq	727	12.1%
3	Packard Bell	711	11.8%
4	IBM	493	8.2%
5	Hewlett-Packard	328	5.4%

New Technotes Give Developers More Options

If you develop for the Mac OS platform, you're probably familiar with the Macintosh Technical Notes, which are technical documents that supplement the *Inside Macintosh* technical documentation. To give you better information in the formats that you prefer, Apple has replaced the Macintosh Technical Notes with a new form of supplementary technical documentation, called (to distinguish them from the old Macintosh Technical Notes) *Technotes*.

Existing Macintosh Technical Notes will continue to be available on the Developer CD Series and the World Wide Web, but new material will appear as Technotes.

Technotes will be available in more locations:

• You can order a yearly subscription to printed Technotes through Field Copy and Printing (for details, call 415-323-3155 or send e-mail to AppleLink address FIELDCOPY or Internet address fieldcopy@applelink.apple.com).

• You can access Technotes through the World Wide Web (at location http://dev.info.apple.com/technotes/Main.html).

Two new things about Technotes are worth noting. First, you can access the Technotes on the World Wide Web through a powerful new search engine. This feature will help you get to the documentation you need as quickly as possible. Second, Apple now welcomes Technote contributions from developers like you. (Technotes are best suited for shorter contributions; *Apple Directions'* sibling publication, *develop* magazine, remains the premiere publication medium for in-depth technical articles of interest to Mac OS developers.) For details on how to contribute a Technote, see the Technote Web page (at the location mentioned earlier).

• You can also find Technotes on the Developer CD Series from Apple; they are collected quarterly on the Reference Library Edition.

In addition, you will be able to read Technotes in one of three formats: Adobe Acrobat, ClarisWorks 4, and QuickView (the file format used by Apple's Macintosh Programmer's Toolbox Assistant).

Chinese Dictation Kit Wins COMDEX Asia "Best of Best" Award

Apple Computer, Inc., recently introduced the Apple Chinese Dictation Kit, a new product that converts Mandarin (Putonghua) speech into simplified or traditional Chinese text. The product, which works on a Chinese-language-equipped Power Macintosh computer with at least 4 MB of free memory, allows users to enter text about five times faster than the most popular keyboard-based input methods. Because of the product's ease of use, product design, and usefulness, it was awarded both the "Best Software Product" and the "Best of the Best" (best overall product) awards at COMDEX Asia in November.

The Apple Chinese Dictation Kit consists of two components: the Apple Chinese Dictation Kit software and a special microphone, called the Apple Dictation Microphone. Users then train the software to recognize their voices by reading several pages of text into the microphone. After the software has analyzed the voice recording and created a user profile file, a user can then dictate to the computer by speaking one phrase at a time.

The Apple Chinese Dictation Kit can recognize approximately 350,000 phrases, and users can add phrases to customize the system's vocabulary. A phrase-adding utility scans text that has already been entered through dictation and adds new words to the system's vocabulary. The kit also provides an error-correction feature: when a user clicks on the first character of an incorrectly entered phrase, the system presents a list of likely alternative phrases.

The Apple Chinese Dictation Kit will be available during the first calendar quarter of 1996 in the United States and Canada, at a price of $299. Internationally, AsiaSoft will be distributing the product for Claris Corporation. For more details, you can reach AsiaSoft at Internet address asiasoft@asiasoft.com, or you can visit the company's World Wide Web page at http://www.asiasoft.com/. ♣

==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==>

==> ==> **APPLE DIRECTIONS EXPRESS** ==> ==>

==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==> ==>

At-a-glance, up-to-the-moment business news from Apple
Apple Directions Express–the Apple Online Business Bulletin
Sent to you instantaneously—and free—over the 'net

Apple Directions Express is the official online business bulletin from Apple Computer, Inc., and the editors of *Apple Directions.* At least every other week, we send you a single, concise e-mail providing you with important, up-to-the-moment news and competitive analysis from inside and outside Apple.

Tired of searching the 'net for Apple-related material? Apple Directions Express not only gives you the news, but points you to Internet locations for complete information about new Apple products and technologies, strategic initiatives, market opportunities, and other subjects that are important to the business of Macintosh development.

SUBSCRIBE now to receive this valuable service. THERE'S NO CHARGE!
Send e-mail to adirections@thing1.info.apple.com.
In the subject field of your message, type the string "subscribe <your real name>".

We'll send you a free pizza over the Internet
if you act NOW!
<well, we would if we could>

Technology

Inside This Section

CD Highlights

System Software and SDK Editions, January 1996

Beginning this month, the product formerly known as the Mac OS Software Developer's Kit CD-ROM becomes the Mac OS SDK Edition of the Developer CD Series. Each quarter, along with the System Software editions in January, April, July, and October, you'll receive a collection of over 30 individual Software Developer's Kits (SDKs) on two CD-ROM discs. These discs provide you with convenient access to vital information for writing software that takes advantage of the services provided by the Macintosh Toolbox.

A typical SDK for a Toolbox service provides you with the following key components:

- system software extensions
- programming interfaces and libraries
- sample code
- technical documentation (supplemental to *Inside Macintosh*)

These are the basic components you'll need in order to understand and use a Toolbox service. (Note, however, that the discs generally will not include special development tools that may rely on specific Toolbox services.) The discs also include an Obsolete/Unsupported folder, which contains components that are provided for historical reference only—do not expect future updates or support for them.

Although we expect the SDK Edition to provide you with a reasonably complete set of SDKs relating to Macintosh system software services, we cannot guarantee that every SDK will be included. Some SDKs may not be appropriate because of licensing restrictions, special distribution requirements, or changes in distribution strategies.

For complete information, see the software license included with the CD-ROM (and stored in electronic form in the Licensing Info folder). Note that this license allows you to distribute with your applications certain libraries and system software extensions included on the CD-ROM.

System Software, January 1996

So, in addition to new versions and localizations of system software 7.5.1 and 7.5.2, QuickDraw 3D, QuickDraw GX, and QuickTime, here's what's new and revised this month.

BBEdit Lite 3.5

BBEdit Lite is a freeware derivative of BBEdit 3.5, the popular and critically acclaimed text editor for programmers, HTML authors, users of online services, and anyone else who needs to edit plain-text files.

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. (BBEdit Lite copyright © 1992–1995 by Bare Bones Software, Inc. All rights reserved.)

Debugging Modern Memory Manager

This package contains a debugging version of the Modern Memory Manager. Once installed on Power Macintosh computers, it allows greater control in detecting and eliminating Memory Manager bugs. The package also provides a control panel that allows you to dynamically enable and disable the debugging features.

Developer Notes Update 01/96

This folder includes the document *PB 190 RAM Card Error Note,* which corrects an error in the developer note for the Macintosh PowerBook 190 computer. The error affects developers of RAM expansion cards for both PowerBook 190 and PowerBook 5300 computers. *PB 190 RAM Card Error Note* describes the error and provides guidelines for developing RAM expansion cards that are compatible with the Macintosh PowerBook 190 and PowerBook 5300 computers.

PopupFuncs 2.6.1

PopupFuncs is a productivity tool for developers. When invoked by clicking a control in a source code window's title bar, PopupFuncs creates a pop-up menu of every function contained in a source code file, allowing you to see instantly the contents of an unfamiliar file. If a name is selected from the menu, the file is immediately scrolled to the beginning of that function. PopupFuncs works with CodeWarrior, MPW, THINK C/Symantec C++, SADE, BBEdit, and QUED/M, and parses C, C++, Pascal, Object Pascal, assembler, Rez, and Fortran

please turn to page 23

OpenDoc Human Interface FAQs

Scripting, Buttons, and Hypertext

By Kerry Ortega and Dave Curbow, OpenDoc Human Interface Team; and Jon Pugh, OpenDoc Engineering

This time we're going to focus on supporting scripting within OpenDoc. This article doesn't address all the questions on scripting, so please send us any others you think of; we'll answer them in future columns.

Q. How do you script actions like selecting a menu and other controls (such as sliders)?
A. This might be a good time to briefly mention the different levels of scripting support. If you want more details on scripting, see Chapter 9 in the *OpenDoc Programmer's Guide.*

First, a part can be scriptable. To be scriptable, your part editor must publish a list of its content objects (for example, lines and words in a text part) and operations (for example, bold in a text part) and then accept semantic events (for example, "Set Data to Bold"). For your part to be fully scriptable, semantic events must be able to invoke any action a user might be able to perform. Someone can then write a script to invoke the particular user action.

The next level of scripting support is for your part to support recording. If your part is recordable, it can capture the user's actions as a series of semantic events, convert the actions to scripts, and replay them at a later time to reenact the actions.

The final level of scripting support is to make your part's interface customizable (sometimes referred to as being "tinkerable"). If your part is customizable, users can invoke all the actions through scripts, and they can change the behavior of the operations. To be customizable, the part editor must be scriptable and must define content objects and operations for interface elements such as menus and buttons.

Getting back to your question, your part's interface must be customizable to support the actions you mention. We've said in a previous column (*Apple Directions,* October 1995) that users may attach a script to a scriptable part—for example, through the Settings dialog box. A script may have multiple sections of script code (handlers) for the different operations whose behavior you want to change. Suppose that you want your part to support events such as mouseDown, mouseUp, mouseEnter, mouseLeave, and pickMenuItem; when designing your part editor, you must define event names for each operation that your editor will support. Before your editor processes any event, the editor must check with the attached script to see if it can handle the event. If so, that section of the script is run.

If you want to allow users to write scripts that control your part, you should publish your list of event names in your part's documentation and 'aete' resource. Because HyperCard uses a set of event names for the common actions a user might perform, you might want to look at HyperCard events as examples to follow. By the way, HyperCard uses the subroutine event to implement many of its events. For more details on subroutines, you may want to look at the Developer Notes document on the AppleScript CD.

Q. What about scripting document-level commands such as Open, Close, and Print? What do I have to do?
A. There are operations that relate to the entire document—for example, closing, saving, and printing the current document, or opening a different document. The OpenDoc shell handles three of the four required events (the fourth, Open Application, does not apply in the OpenDoc environment). Your part editor can override the definitions of these events. Note that definitions of these events are slightly different than in today's applications:

• Quit—close all windows of the document
• Print—print the active window
• Open—open the specified document

The OpenDoc shell also supports three of the core events, which your editor may override. The events are

• Close—close the active window
• Save—save the current document
• Save As—save a copy of the current document into a new document

The OpenDoc shell will first determine if the editor of the root part implements these commands; otherwise, the shell provides the default implementation.

In addition, if the user wishes to attach a script that is to be executed for one of these events, they must be attached to the root part of the document.

Q. How should a script identify parts within a document?
A. This can be very tricky for any document that can be revised, because parts need not be named or have unique names. Furthermore, the index of a part may vary over time, and different parts will have different ways of indexing their contents. For example, "line 3" is fairly obvious; "circle 3" is not. Therefore, the only reliable way to refer to a part is by its ID. Unfortunately, the ID is not a very user-friendly or visible number; however, the user can always find the ID of a part through the Part Info dialog box.

This problem isn't unique to OpenDoc, or even to AppleScript. There just doesn't seem to be a great solution to this problem.

Q. In a previous column, you said that parts should place their scripts in the Settings dialog box. Doesn't that force parts to provide some text-editing capability to edit scripts?
A. Yes—to allow users to edit the script, your part needs to provide some mechanism for viewing and editing the script. For a text-based language, that means you need to support text editing. (Remember that some script languages may have a representation other than text.) The text editor you provide can just be a simple multiline edit control. For the future, we are looking at providing a scripting service that parts could use.

Q. I want to create a "button" that really is an embedded graphic in a container part—like having a picture in a HyperCard stack. When the user clicks the "button," the container part will run some script. In other words, the graphic part does not support scripting, but its container provides the scripting for the embedded part and gives it button-like behavior. Any suggestions?
A. The basic rule in OpenDoc is that a click goes through to the smallest part containing the mouse pointer. So, if the user clicks on the graphic, that part would get the click. But in your

example you want the container to get the click and handle it. To do this you need to "fake out" the embedded part so that it doesn't get the click, but the container does.

There is one easy way to get around this problem. If a part is "bundled," then it doesn't get the click—its container does. So, the container could always set the bundled checkbox (in the Part Info dialog box) for all parts that are embedded within that container.

Whenever the user clicks the graphic part, the container part would get the click, determine which "button" was clicked, and then run the appropriate script.

This all makes sense when the part is in "run" mode. However, you need to allow the user to edit the buttons as well. If you have "run" and "edit" modes, you should turn the bundle property off—otherwise, the user can't edit the picture of the button.

Important: Our guidelines recommend that the user should be the only one who turns the bundled checkbox on or off. But this example is a case in which the container part could set the bundle property.

Q. How does one avoid modes when editing buttons?
A. Modes aren't always a bad thing—we work in different modes all the time. Modes are bad when they aren't clearly visible. So, a mode for editing a script, editing the name of a button, or resizing a part should be clearly distinct from other modes in which the user could be working.

Attributes such as button name, size, and color are properties of the button, and you can place them in the Settings dialog box. Even though setting attributes is a mode, the mode is clearly visible and distinct.

Sometimes your users may need "edit" and "run" modes. For example, imagine using a part builder to make frequent and numerous property changes. If the part required you to bring up the Settings dialog box every time you wanted to add or change a part, after awhile you'd probably want an easier and faster way to enter these changes directly—an "edit" mode. These builder applications really need the two modes, because the modes really reflect two different user tasks. They are long-term modes, and as long as they are clearly visible and distinct from one another, using them should be clear to the user.

Q. How is hypertext achieved in OpenDoc? I see a couple of ways to do this: Embed a button in the text part or make hypertext a "style." Which method would you suggest?
A. In the 1.0 release of OpenDoc, we have no specific support for hypertext links. We know this is an important issue and plan to work on it for a future release. Right now, any approach will have the same major problem: namely, OpenDoc does not provide any mechanism to display any destination of a hypertext link smaller than a part. For example, there is no way to create a link to a particular paragraph in a text part, or a particular portion of a picture.

We are interested in hearing any suggestions you have. ♣

Demystifying DSOM

How DSOM Is Connected to OpenDoc, and Why You Should Care

By Gregg Williams,
Apple Directions *staff*

As a Mac OS developer, you're most likely a pretty clever individual, but you probably don't get away from the desktop very much—that is, you regularly create whiz-bang stand-alone Mac OS applications, but you may not know much about creating applications that reach out over the network. And that's OK—for now.

However, things are changing. If you believe books like *The Essential Distributed Objects Survival Guide* (see the "Resources" box on page 23 for details), in the next five years, stand-alone desktop applications are going to become as quaint as the all-in-one home stereo systems of the 1960s. Software that exists across a worldwide network, according to this book and other sources, will be the Next Big Thing and will be more powerful than software that is isolated on one computer.

Some would say this prediction is just another case of silicon snake oil. Sure, we have client/server software, but it enjoys only occasional use, and it hasn't changed the world, especially the personal computer desktop. But the proponents of distributed objects say that this will change because of two things: software objects, and standards that allow them to communicate across networks regardless of how they were designed or what kind of computer they live on.

So what does distributed-object technology have to do with you? You may be using it in the not-so-distant future so you need to know about it today, because you need to make short- to medium-term decisions that will leave you poised to take advantage of this new technology when the time comes. In particular, there is a hidden connection between this new technology—in the form of Distributed SOM (also called DSOM)—and a technology that will, guaranteed, become more commonplace on the Mac OS platform in the next year—namely, OpenDoc.

As is so often the case, DSOM is a concept that sits at the top of a pyramid of concepts, and you have to understand the foundation concepts first: CORBA, SOM, and the mechanics of using SOM within OpenDoc. I'll motivate the "building" of this pyramid with a series of questions.

What Is CORBA?

In 1989, a number of companies formed a consortium called the Object Management Group (OMG); its purpose was to create industry standards for the not-yet-created commercial object-oriented software systems that were sure to be created. (The OMG has grown since then; it currently has over 500 computer and communications companies as members.)

The first specification that the OMG defined was for what it called an ORB (Object Request Broker). An ORB would provide a mechanism through which object-oriented software from different vendors could communicate with each other across a network, even when the objects ran on vastly different computers.

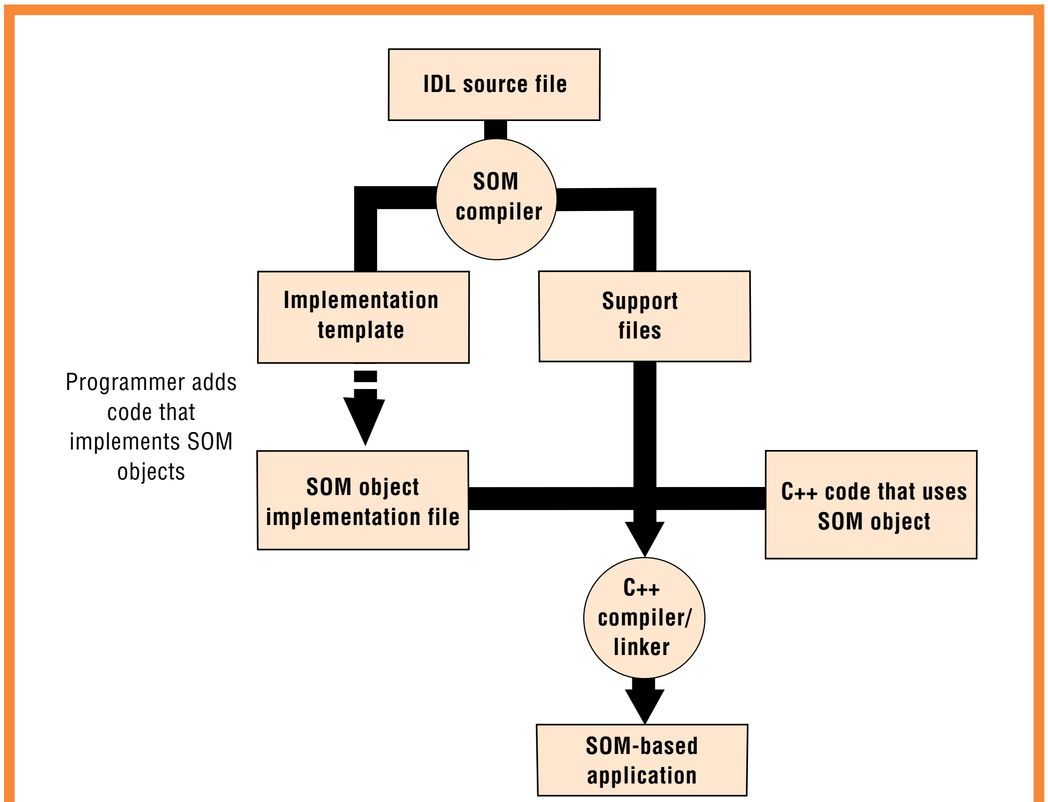
In September 1991, the OMG selected a standard interface that implemented the mechanisms of an ORB. Called CORBA (Common Object Request Broker Architecture), this architecture uses a declarative Interface Definition Language (IDL) to specify the interface of an object (including its methods and its attributes) independent of the particular programming language and the source code used to implement the object.

What Is SOM?

SOM (System Object Model) is a software technology created by IBM for developing and packaging object-oriented software. SOM makes real many of the advantages promised by object-oriented programming. It offers a number of advantages over current object-oriented technologies:

- It allows you to create object-oriented software using, potentially, any programming language. (SOM currently supports C and C++, with support for Smalltalk promised. IBM recently announced SOM support for (gasp!) object-oriented COBOL in the form of its VisualAge for COBOL for OS/2, AIX, and MVS products.)
- It is cross-platform (currently running on Mac OS, Windows 3.x, Windows 95, and OS/2 systems, as well as IBM's AIX version of UNIX, and the IBM MVS mainframe and OS/400 minicomputer operating systems).
- It makes software reuse a reality. You can create objects in one language and call them from another language. (In contrast, object-oriented software created today with one C++ compiler cannot even be used by software created by a different C++ compiler.)
- Other people can use the object-oriented software you create without you having to give them your source code. (You give them object code instead.)
- If you do it properly, you can improve or bug-fix a class library and replace the old version with the new without complications. Software that uses this class library continues to work without having to be recompiled. (Currently, most nontrivial changes to a C++ class library force you to recompile any software that uses the class library, which decreases the ease of use that object-oriented programming promises.)

One important thing to remember is that SOM is a CORBA-compliant ORB. In terms somewhat closer to plain English, this means that any software objects created with SOM will work with any other CORBA-compliant objects, even objects created using tools from other vendors and (if certain conditions are met) objects running on computers (across a network) that use different processors and operating systems.


IDL source file
SOM compiler
Implementation template
Support files
Programmer adds code that implements SOM objects
SOM object implementation file
C++ code that uses SOM object
C++ compiler/ linker
SOM-based application

Creating SOM-based software. Most of the programmers' work occurs when they fill out the implementation template with code that implements the SOM object. Direct to SOM compilers will make the development of SOM objects much easier than this diagram shows.

How Do You Create Software Using SOM?

To achieve the language independence and compiler independence that SOM promises, you must create your SOM classes as shown in the figure "Creating SOM-based software." (This figure shows how you create SOM classes if you are using C++; the process is similar for any other language that supports SOM.)

First, regardless of what language you will be using, you define the new SOM class in IDL (Interface Definition Language), the syntax of which is very similar to that of C++; the result is called an *IDL file.* (In the future, this process will be much simplified by "direct-to-SOM" compilers, which essentially merge the SOM and C++ compilers. With a direct-to-SOM compiler, you will be able to write C++ classes, and the compiler will produce SOM objects—a definite improvement over having to create IDL files.)

An IDL file uses C++-like statements to do the following:

- name the class being defined
- specify the new class's parent classes
- name the *attributes* associated with the class (in SOM, an attribute is an externally accessible variable)
- name the *methods* associated with the class (a method is a procedure that an object executes when it receives a message telling it to execute that method)
- specify some housekeeping code, including a list that specifies the order in which the class's methods will be referenced internally

Second, you must use a SOM compiler to generate a series of files that the target language (in this case, C++) needs for generating SOM-based software. One of these files is called an *implementation template.* (To simplify this explanation of how SOM works, I'll refer to two other files that the C++ compiler needs as *support files.*)

Third, you must add the code that implements all the methods defined in the IDL file. The implementation template just created contains "empty" methods, one for each method defined in the IDL file. In this step, you modify the implementation template by adding the code that implements the class's methods. The resulting file is called the SOM object implementation file.

Fourth, you write the "main" code of your application; this is code that uses the SOM class to create instances of that class (that is, to create objects that instantiate that class) and to invoke methods on objects. The support files (mentioned earlier) that are part of the compilation process allow the computer language being used to invoke a method on an object using a simple procedure call.

For example, if the IDL file defines a class named Car that has a method named "accelerate," you would create a Car object in C++ with the statement

```
myCar = new Car;
```

and tell myCar to accelerate with the statement

```
myCar->accelerate(ev);
```

(The variable ev points to an environment structure that every SOM method call must include.)

In contrast, the equivalent C code to create the same car and accelerate it would be somewhat different but would achieve the same effect:

```
myCar = CarNew();
_accelerate(myCar,ev);
```

Getting back to the process of creating SOM-based software, the final step is to take the SOM object implementation file, the support files, and the main C++ program, and use a C++ compiler and linker to create an executable program.

Admittedly, I've left out a few details in both this explanation and the corresponding diagram to emphasize the overall structure of how you create and use SOM classes, but you can get the details from a number of sources. One such source is Chapter 2 of *Object-Oriented Programming Using SOM and DSOM* (see the "Resources" box for details).

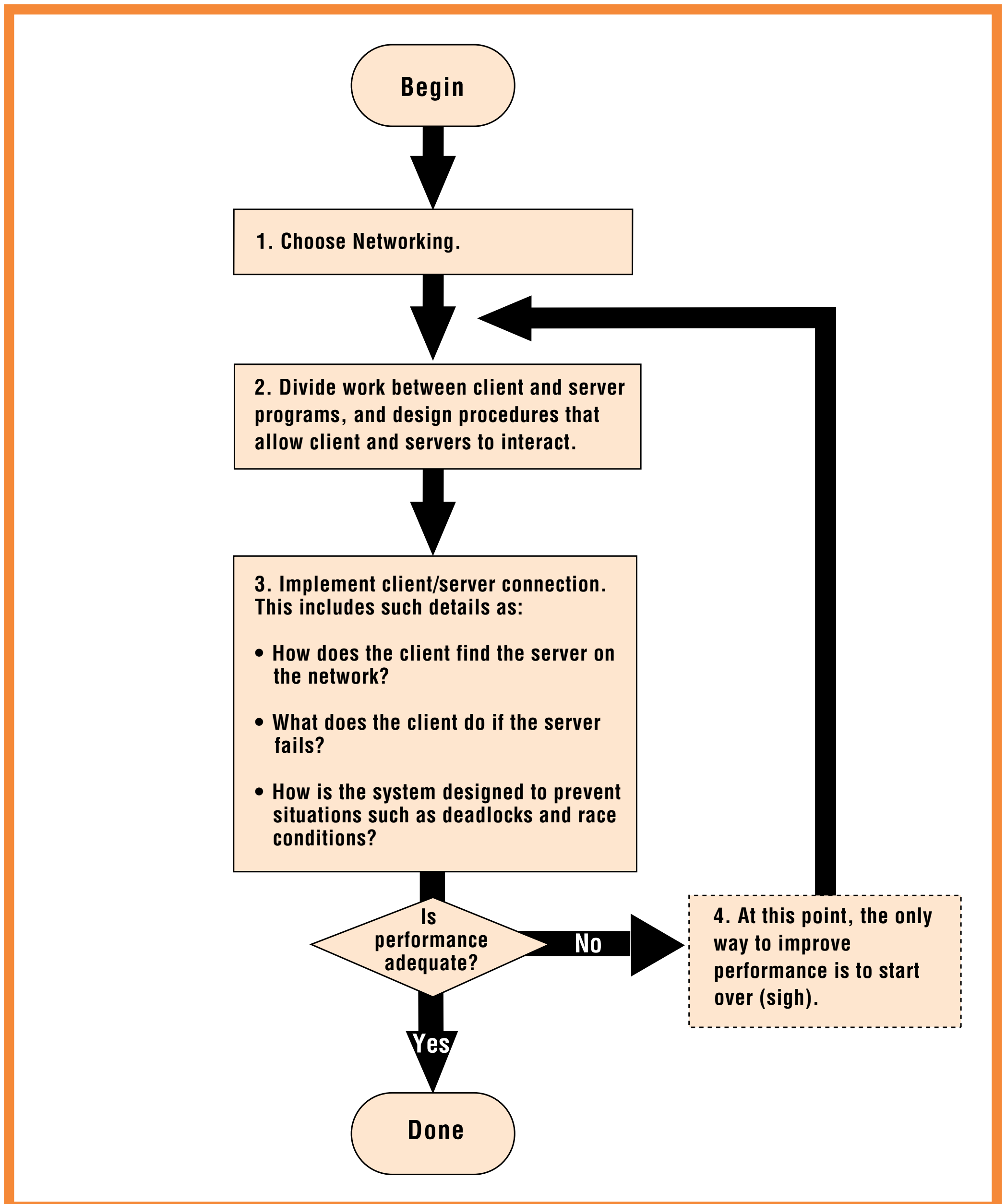
Why Does OpenDoc Use SOM?

Quite frankly, SOM contributes greatly to making OpenDoc component software usable in the real world. Without SOM, all OpenDoc parts would have to be created using the same C++ compiler. Also, without SOM, modifying and recompiling a class library would force the recompilation of all of its subclasses (which would, in some situations, complicate the distribution and bug fixing of OpenDoc parts and other SOM software).

In addition, SOM provides dynamic binding for its objects. OpenDoc must have this feature for users to be able to add new parts to an existing document, and SOM provides a ready-made implementation of dynamic binding.

How Does SOM Enter Into the Creation of OpenDoc Parts?

OpenDoc is a component software architecture implemented as


Begin
1. Choose Networking.
2. Divide work between client and server programs, and design procedures that allow client and servers to interact.
3. Implement client/server connection. This includes such details as:
• How does the client find the server on the network?
• What does the client do if the server fails?
• How is the system designed to prevent situations such as deadlocks and race conditions?
Is performance adequate?
No
4. At this point, the only way to improve performance is to start over (sigh).
Yes
Done

Conventional client/server design. If you don't get it right the first time, you've basically got to start over.

a set of CORBA-compliant class libraries. To create an OpenDoc part from scratch, you must subclass the ODPart class that is implemented in the OpenDoc shared library. You do this by writing an IDL file that defines your part, compiling the IDL file with the SOM compiler, adding source code to the resulting implementation template, and compiling and linking the resulting file, as shown in the "Creating SOM-based software" figure (page 18).

Actually, there are better ways to create OpenDoc parts. One tool from Apple Computer, Inc.—PartMaker—takes your name for a part and creates a series of source files that you then modify to create your OpenDoc part. In addition, the OpenDoc Development Framework (ODF) is a software framework that gives you even more help in creating OpenDoc parts—and will help you create equivalent parts for the Microsoft Windows platform.

What Is DSOM?

Finally, with all the underlying concepts explained, I can begin to explain what DSOM is, how it relates to OpenDoc, and why you should care.

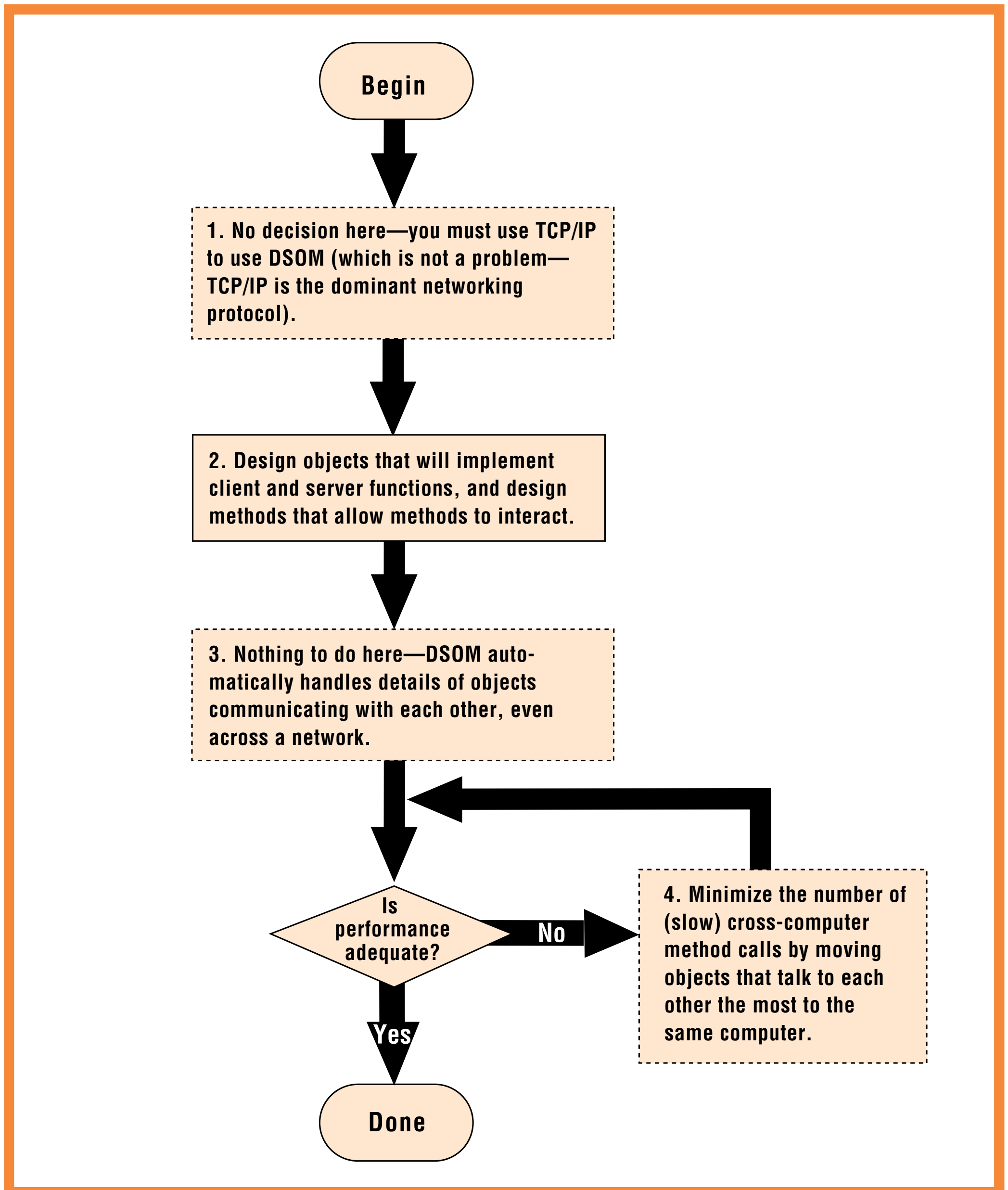
Distributed SOM (DSOM) is a component of SOM (added in IBM's release 2.0 but not yet available on the Mac OS version, which is numbered 2.0.7) that supports access to objects across a network. Code properly written to use DSOM will access an object the same way—as if it were remote—regardless of its actual location.

Two pieces of the DSOM architecture, the Interface Repository and the Implementation Repository, make DSOM work. The Interface Repository contains information about SOM classes and what methods and attributes they support. The Implementation Repository knows the name of the server on which an object resides. On a computer that is running DSOM, when a program invokes a method on an object, DSOM uses both the Interface and Implementation Repositories to determine where the object resides, to cause the method to be invoked on the object (wherever it is, on the same computer or across the network), and to return the result to the program that originally invoked the method.

Herein lies the beauty of DSOM. If you understand how DSOM works, you can write SOM objects today that can easily be modified to become distributed objects when DSOM becomes available. (One issue that must be addressed is the fact that distributed objects should use standard remote programming techniques—that is, asynchronous messaging—to ensure that message transmission delays and errors do not cause an individual software object to "lock up" the computer on which it is running.) Once your SOM objects have been converted to DSOM objects, you can place them on remote computers (which must also be running DSOM) and the software that calls them will continue to work, whether the called object is located locally or remotely.

Why Might I Use DSOM?

DSOM tremendously simplifies the process of writing software that works across a network—this includes but (as you will later see) is not limited to groupware and real-time collaboration software and any kind of client/server


Begin
1. No decision here—you must use TCP/IP to use DSOM (which is not a problem—TCP/IP is the dominant networking protocol).
2. Design objects that will implement client and server functions, and design methods that allow methods to interact.
3. Nothing to do here—DSOM automatically handles details of objects communicating with each other, even across a network.
Is performance adequate?
No
Yes
4. Minimize the number of (slow) cross-computer method calls by moving objects that talk to each other the most to the same computer.
Done

Client/server using DSOM. Not only does the use of DSOM eliminate the problems of implementing network connections, it also makes it possible for you to improve performance without redesigning major portions of your solution.

software. DSOM isolates you from many tedious and difficult networking issues. Because DSOM is cross-platform, the software objects you create will be able to communicate with each other, even if they are executing on computers that use different processors and operating systems.

In addition, if you are already using SOM (which you will be when you create OpenDoc parts), you will be able to use the software objects inside those parts in a distributed environment with minimal extra work. (An example at the end of this article shows you how easy it is to do this, and why you would want to.)

Conventional Client/Server Design

First of all, a word up front: DSOM doesn't really make new kinds of network-based solutions possible, but it does make them much easier to implement. Practically speaking, though, if the difference is big enough, you *will* attempt things with the new technology that you formerly wouldn't have—so, in a way, DSOM does make new kinds of network-based solutions possible.

Two figures, "Conventional client/server design" (page 19) and "Client/server using DSOM" (page 20), show exactly how and where DSOM makes a difference. (If your eyes glaze over at the term *client/server,* just remember we're talking about a software solution that can benefit from computing power or data on multiple computers across a network—which covers a lot of things you may want to do.)

Looking at "Conventional client/server design" first, note that there are three main steps involved in creating a solution:

- Step 1: Decide which networking protocol to use.
- Step 2: Divide the work between the client and server computers and write the code that implements each computer's share.
- Step 3: Write all the code that implements the communication between the client and server programs.

According to the people I talked to, step 3 is an extremely complicated and tedious process. In the words of OpenDoc architect Kurt Piersol, step 3 will involve "months and months of work for your best programmers." In addition, you may have to hire consultants who specialize in this arcane field; according to Piersol, this is not work you can assign to your average C++ programmer.

If the resulting software is too sluggish, this is your only recourse:

- Step 4: Analyze the current system's performance (which is probably being slowed down by large volumes of network traffic) and start over again.

How DSOM Improves Client/Server Design

Contrast this with the approach shown in the "Client/server using DSOM" figure (page 20).

First, there is no step 1, really. If you want to use DSOM, you will probably be forced to use the TCP/IP networking protocol (although some platforms support other protocols, such as Netware's IPX and IBM's NetBIOS). However, since TCP/IP is the most popular networking protocol available (can you say "Internet," boys and girls?), this is usually not a problem. In any case, the transport mechanism underlying DSOM is not set in stone, so you may be able to use other transport mechanisms in the future. And if that happens, guess what? DSOM objects you've already created will continue to work!

Step 2 for DSOM is the same as for conventional design: You must divide the work up and write the procedures that implement this work. But because DSOM is object-oriented, you think in terms of software objects, not procedures.

The first big win of DSOM is in step 3—there's *nothing* to do here, and step 3 is the most difficult step of conventional client/server design. DSOM takes care of all the network communications, and it probably does a better job than you would be able to.

The second big win of DSOM occurs if the resulting system doesn't perform as well as you'd like. When you find objects that are generating inordinate amounts of network traffic, you can "balance the load" by moving these objects to the same computer. This allows you to redesign your client/server solution in a matter of hours or days, not months.

Once you get a conventional client/server solution running, you have a solution, but it's a "hard-wired," inflexible solution. A DSOM solution, on the other hand, has a lot of implementation independence built into it, which means that if your solution needs to be changed, DSOM gives you more options for doing so.

Using DSOM to Enhance an OpenDoc Part

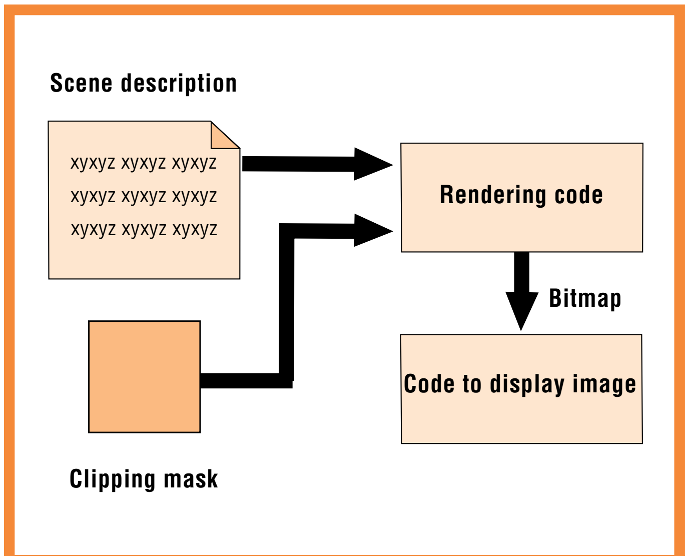
Here's another way of looking at DSOM: *If you're already using SOM, then it costs you very little more to use DSOM, and you get all the benefits of DSOM for close to free.*

Guess what? If you're using OpenDoc (and you *are* planning on using OpenDoc, aren't you?), you're using SOM. And that means that, once DSOM becomes available, you'll get all its benefits for a small amount of extra work.

But you may say that you aren't doing groupware or client/server solutions, and therefore DSOM doesn't offer any advantages to you. Here's a counterexample that demonstrates how useful DSOM may be to you as an OpenDoc developer, even if you aren't interested in groupware or client/server solutions:

Let's say you've built a rendering part for OpenDoc. Such a part would contain a three-dimensional scene defined in some sort of 3D modeling language and would display that scene by rendering the image into a displayable bitmap. The part, internally, would have

- the scene description
- a clipping mask
- code that renders the scene
- code that displays the rendered bitmap



An OpenDoc rendering part. Contrast this single-computer solution with the figure diagrammed in "Distributed rendering using DSOM."

The figure "An OpenDoc rendering part" (page 21) describes this part at a block-diagram level. You could have designed this part differently, but you knew DSOM was going to be available in a few years, so you designed it this way (for reasons that will become obvious).

Time passes, and—boom!—DSOM is now available. Just in time, too—your customers are rendering insanely complex graphics and are complaining about how slow your rendering part is. Fortunately for you, your most vocal customers think that distributed rendering (that is, rendering different parts of the image on different computers and combining them on one computer) is an acceptable solution.

Here's how much (or, actually, how little) work it is to change a rendering part into a distributed rendering part:

• First, take your core rendering code and modify it to make it into a DSOM object that does rendering using asynchronous messaging. This object must include a render-this-for-me method that accepts a scene description and a clipping mask and returns the specified part of the rendered image as a bitmap.

• Second, place distributed-rendering objects on various other computers (or maybe servers) on your client's network. You can also place a distributed-rendering object on the computer that hosts the "master" rendering part.

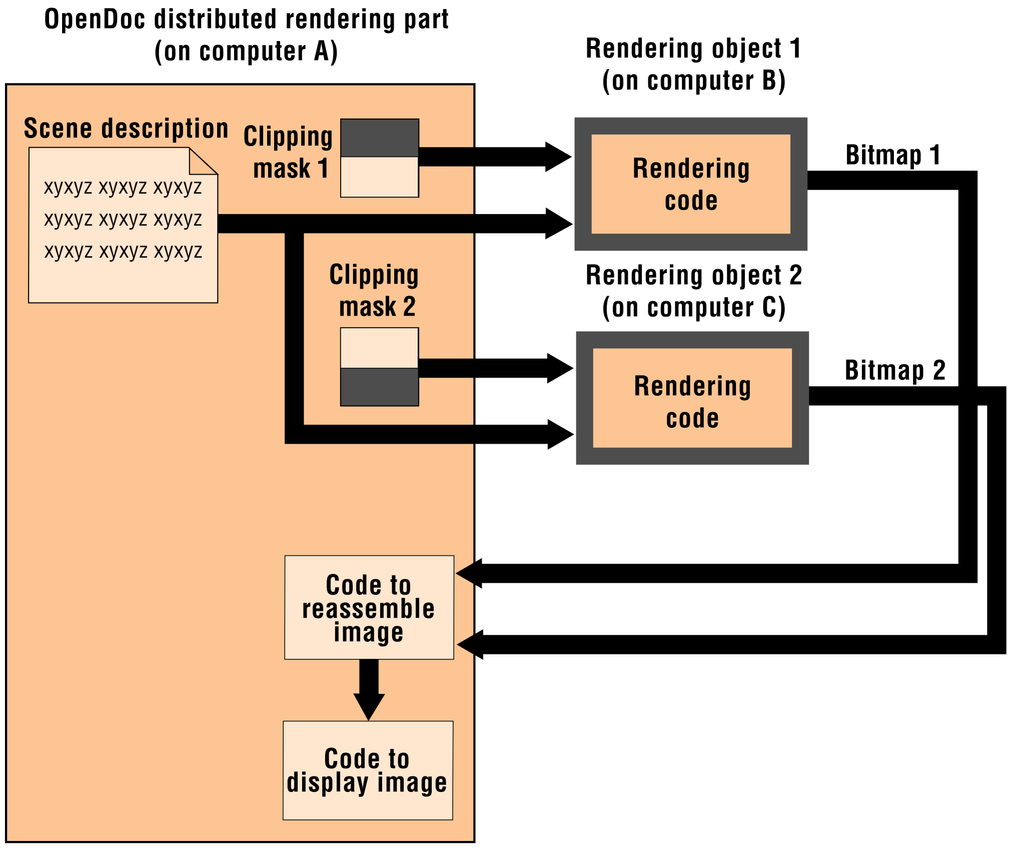
• Third, modify your original rendering part to farm the rendering work out to the available distributed-rendering objects. The code that calls the render-this-for-me method is the same regardless of whether the object being addressed is on the same computer or across the network.

• Fourth, write some code that combines the individual image pieces into the desired final image, then feed it to the existing, unchanged code that displays the image. And you're done!

What does the end result look like? The figure "Distributed rendering using DSOM" (below) shows how the resulting distributed-rendering part works with the distributed-rendering objects. Compare this with the other figure ("An OpenDoc rendering part" on page 21) to see how much similarity there is between the two approaches.

Conclusions

I hope this article has helped you understand how SOM fits into OpenDoc and how DSOM works.



Distributed rendering using DSOM. By repackaging existing software (in this case, the rendering code block from the figure "An OpenDoc rendering part" on page 21, a single-computer rendering part has become a distributed rendering part.

Beyond the high-level overviews of these technologies, though, I hope you will remember two important ideas from this article. First, realize that the step from OpenDoc/SOM to DSOM can be quite small, and that the benefits can be quite large. Second, remember that you don't have to be doing groupware or client/server solutions to benefit from DSOM (though it is also very good for those solution areas). *Distributed computing offers a way of bringing more computing power to any problem (who has too much computing power these days?), and DSOM objects allow you to to harness that power without getting bogged down in networking details.*

As I said earlier in this article, don't be misled into thinking, just because DSOM doesn't exist on the Mac OS platform today, that you don't need to plan for it—it will exist at some future date. And, as you well know, your software isn't designed in a day. If nothing else, you need to ask yourself one question: When distributed computing becomes commonplace (probably in the next three to five years), is the software I'm building and designing today on a path to make use of distributed software when the time comes? ♣

Resources

• *The Essential Distributed Objects Survival Guide,* by Robert Orfali, Dan Harkey, and Jeri Edwards (Wiley, 1996). This is a very comprehensive book about every significant distributed-object system known as of early 1995. It devotes five chapters each to the mechanics of OpenDoc and, on the other side of the fence, Microsoft's Object Linking and Embedding (OLE).

• *Object-Oriented Programming Using SOM and DSOM,* by Christina Lau (Wiley, 1995). This book gives a pure explanation of how SOM and DSOM work for creating software objects, but it doesn't cover OpenDoc.

• "OpenDoc Programming Made Easy," by Dave Bice, in *Apple Directions,* November 1994, page 16. This article gives an overview of PartMaker.

• The Announcing OpenDoc 1.0 World Wide Web home page, at location http://www.opendoc.apple.com, gives you access to the OpenDoc 1.0 for Mac OS software, plenty of documentation, and the latest developer release of the OpenDoc Development Framework.

• "SOMobjects: A Practical Introduction to SOM and DSOM" (document number GG24-4357-00, available from IBM).

• IBM's object-technology World Wide Web site, at location http://www.austin.ibm.com/developer/objects/object_tech.html.

CD Highlights

continued from page 15

source files. (See the file PopupFuncs Notes.c for details.)

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.

Sample Code Update 01/96

This package contains five code examples, which will be rolled into the Sample Code folder on the February 1996 Tool Chest edition of the Developer CD Series.

• *7Edit 3.1.* This application code gives an example of creating a scriptable application—that is, an application that goes far beyond supporting just the basic four events of the Required suite of Apple events (Open Application, Open Documents, Print Documents, and Quit Application).

This latest version of 7Edit is similar to the Scriptable Text Editor. Although it may not be as fully scriptable as the Scriptable Text Editor, it should give you a good idea of how to make your application scriptable. In particular, it shows you how to handle the "whose" clause in AppleScript. This version of 7Edit also demonstrates QuickDraw GX printing and Drag Manager support.

• *ChromaKeyMovie.* This simple application shows alternative approaches to removing a color from a QuickTime movie while it is playing and allowing a separate image to be displayed in the removed regions. It demonstrates several features available in QuickTime and Color QuickDraw.

• *Fragment Tool.* This simple application allows basic manipulation of code fragments. It lets you combine or separate several code fragments and view and edit various pieces of information associated with each code fragment.

• *MenuScripter 3.1.* This example shows how to send data to subroutines in your AppleScript scripts. By constructing Apple events that are sent to a script handler in a script using OSAExecuteEvent, OSADoEvent, and AESend, the example shows how to call subroutine handlers in a script and send positional or labeled parameters to that subroutine. This example is derived from the MenuScripter application, which was created to demonstrate use of the Open Scripting Architecture, through which users can change scripts associated with menu commands.

• *Show Movie.* This is a small application designed to load and play movies. It demonstrates several useful features in QuickTime and ways to use them.

Coming Next Month

Next month's column will feature a trip report from the 1995 European Developer Forum.

Alex Dosher
Developer CD Leader

Human Interface

Holiday Magic

By Peter Bickford

It's time once again for the annual holiday edition of the Human Interface column, wherein I cart out my human interface wish list for the coming year. This gives me a chance to periodically exorcise all my petty complaints about interface design instead of letting them bottle up inside me until they reach dangerous proportions. Without this wish list, I'm pretty sure I'd turn into a bitter old man who wanders the streets of Cupertino staring at the ground while cursing under his breath about sloppy Windows ports and cryptic Command-key combinations.

The trouble is that any proper holiday wish list should contain both little items that are easy to get, and a few big-ticket items just in case Santa thought you were extra good this year. You know, "I want a Power Rangers pencil case, a G.I. Joe with Kung Fu grip, and a peaceful solution to the war in Bosnia." It's easy to come up with the list of little wishes (and by the way, thanks to all the developers in previous years who curved their quotation marks, rearranged their dialog buttons into the proper order, and revised their black-on-dark-gray color schemes). The real challenge is coming up with the big things—what do I really want to see in tomorrow's software? Strangely enough, the answer came to me this weekend as I was trying to electrocute myself.

But First . . . the List

Before I get on to that story, let me first give you this year's holiday wish list. Remember, although I'd be delighted to get everything here, you can make this a happier holiday season for me and your customers with any one of these items.

1. Someone to do the housekeeping—or at least set the clocks. When I became a homeowner this year, I started to realize that there are a whole series of rituals you have to learn if you want to keep things running right: mowing the lawn every couple of weeks, pruning back the roses, changing the filter in the furnace, and so on. When you add these chores to the countless others that go into life, it starts taking up quite a bit of time. Computers demand similar amounts of attention, and experienced Macintosh owners have developed a practice of periodically running backups, diagnosing and defragmenting their hard disks, rebuilding their desktops, and so on. Otherwise, problems accumulate and Bad Things start to happen spontaneously.

What our computer systems need are the equivalents of the frost-free refrigerator and the household staff. Like the frost-free refrigerator, software components ought to be more self-maintaining. Applications should be able to resolve basic problems without intervention from the user. When a preferences file is missing, an application should create a new one "on the fly." Similarly, when a program that relies on other files is moved to another hard disk, it should try to resolve its links before asking the user for help. For instance, whenever I move my AppleLink folder to another hard drive, it should try substituting its new path when looking for the connection file before complaining of an error.

It would also be great if there were some facility inside the computer whose job it was to keep things running smoothly: fetching your mail, optimizing your hard disk, and so on. We're making some progress in this direction, but the services that have been created so far need to be much less obtrusive and much more reliable before people will begin to trust them. As we begin to move toward agents and other automatic facilities, we need to keep in mind that doing a simple thing well is better than doing a complex thing poorly. Instead of omnisciently trying to optimize my desktop according to an analysis of my working style, an agent might start by automatically setting the clock when daylight-saving time rolls around (after all, the system knows my location from the Map control panel and my Daylight Savings Time setting from the Date & Time panel).

2. An end to gratuitous tool bars. The winner for most overused interface element goes to the tool bar. It seems like some form of mass hysteria is sweeping the world, convincing developers that their applications are just not "on the cutting edge" if they don't have a large collection of indecipherable icons taking up valuable screen space just below the menu bar. No doubt this is the same sort of groupthink that gave us tail fins on cars and had the children of the 1960s going through high school in bell-bottom pants.

The problem is not so much that tool bars are bad in themselves, but that designers are starting to automatically assume that an application needs one. I once met with a group of designers who were showing me some sketches of their interface. Right below the menu bar was a big section filled with squares that they used to represent a tool bar.

"Which functions go there?" I asked.

"We don't know yet, but we'll think of something," they replied.

If you don't know why you need a tool bar—or any interface element—don't use it. Tool bars should only be used in those extraordinary cases when you have a few high-frequency functions that are so useful to have at hand that it's worth permanently taking up a rather large amount of screen space. For most programs, you'd be better served by palettes (which are generally smaller as well as movable), or simply leaving your tool bar functions as menu commands.

3. Some really great games for the Macintosh computer. Oh, wait—what am I thinking? There's Wolfenstein 3-D, Full Throttle, Loony Labyrinth, Troubled Souls, You Don't Know Jack, Marathon. . . . Er, strike this point, just keep 'em coming!

4. For every developer to staff the company's technical support lines for a day. Better yet, have your company set up a special training course in which real users coach you while you do their jobs for a day. One of the problems with being a developer is that we're always taking guesses at what our users are going to be doing and thinking. At our best, we try out designs and prototypes on sample users, but we're usually isolated from the effects of having guessed wrong. The technical support folks, on the other hand, get an earful every time we concoct a confusing error message or convoluted feature. A couple of years ago I sat in on the Apple Computer, Inc., support line and got a real education in how basic aspects of our system software that I had taken for granted (fonts, memory, MultiFinder, and so on) were the basis for most of our trouble calls. With at least some of these areas, all it took was a small change to eliminate hundreds of support calls. Still, you can't fix a problem until you know about it—and there's no substitute for firsthand experience.

Of Circuit Breakers and Magic

But what do I really want most of all from developers? Let me tell you a story. . . .

In California, most houses don't have basements. So when my wife and I were looking for our first home, she would check out kitchens and closets, and I would go straight for the garage. That was where I was going to fulfill my boyhood dream and build a music studio of my very own.

I drew up the plans on my PowerBook, warned all the neighbors, and when my parents came out to visit a couple of weeks ago, I decided that I would either take them to the Tech Museum again, or make them help me lay the foundation. For the rest of the weekend, my dad (who is legally blind) and I shared a powerful father-son bonding experience as we blasted nails into the concrete using a .22 caliber nail gun. ("Dad, put on that ear protection or I'm taking it away from you!" [Blam!] "Hey! I mean it!")

By the end of last week, the framing was up and it was time to do the electrical work. I'd wired any number of outlets and switches in my life, but frankly the thought of going into a 200-amp breaker box to install a new circuit for the studio left me just a wee bit nervous. Like most novice computer users, I was convinced that if I touched it, I would break it. And if I broke it, Bad Things would happen. Two scenarios kept playing themselves out in my head: In one, I was the victorious god of Home Improvement, whose new studio was powered by all the electricity it would ever need. In another, my house was burning down and I was being rushed to the hospital while still clutching a melted screwdriver.

After quizzing all my electrically savvy friends, reading and rereading the safety warnings, and throwing the main breakers, I worked up the courage to unscrew the box and take a look inside. Thanks to standardization and consistency, it turned out that installing a new circuit was dead simple. The new circuit breaker only fit in one way, white wires ran with white wires, ground wires ran with ground wires, and so forth. [Disclaimer: A smarter person would have called an electrician—do not try this at home.] Twenty minutes later, the new circuit was wired up and I was ready to turn it on. When I flipped the breakers back on and got power on the new circuit, it felt like I'd discovered some new world. A few minutes beforehand, the breaker box was the scariest thing in my house. Now I felt I not only understood it, but I could use it to do things that were impossible otherwise. It was magic.

At our best, people get the same feeling the first time they use a Macintosh. At our best, the Macintosh takes the secret (and scary) world of computers and turns it into something that lets average people do things they never could do before. It lets musicians score pieces that are impossible to play alone. It lets entrepreneurs, as well as Fortune 500 companies, manage accounts and projects. It lets a salesman with no artistic experience or staff put together a presentation that dazzles a prospective client.

We're in the club. We're no more impressed by any of these feats than an electrician would be impressed by my installing a simple circuit. But if we really want to, we can do more than just practice our trade, write solid code and earn respect among our peers. We can get into our user's shoes and show them how using a computer can make their life better. We can lower barriers like configuration problems, technical jargon, and arcane feature sets that keep people from using computers to their full potential. We're in the club. But at our best, we can create the products that let people who aren't in the club make magic.

Happy Holidays,
Doc

Peter Bickford is a member of Apple Computer's Human Interface Design Center. He can be reached by AppleLink at THE.DOKTOR or on the Internet at the.doktor@applelink.apple.com.

Business

Marketing Feature

Ideas for Maximizing Installed Base Sales

By Ray Kaupp, President, User Group Connection

Everyone these days is talking about staying close to the customer. It seems like a good idea, so "Mom and apple pie." You gain a better understanding of the features that your customers want, and they recommend your products to others.

Let me suggest a more mercenary, bottom-line rationale. An important reason to focus your marketing attention on your installed base is that they represent the ultimate low-hanging fruit in your marketing orchard. Several studies have shown that the cost of selling to a current customer is a fraction of the cost of winning over a new prospect, perhaps as low as 20 percent. And with a little up-front planning, you can significantly increase your ongoing revenue stream.

While working at User Group Connection, I've been involved in dozens of installed base campaigns. In this article, I share some strategies that have been used successfully by other companies to build customer loyalty and maximize installed base sales.

The Name Game

This summer, my refrigerator sounded its death knell. I had no receipts or warranty documents. All I knew was that I had bought it from Sears a few years ago, so I called their toll-free number. With just my name to go on, the customer service person was able to immediately locate a purchase record that included the model number, price paid, purchase date, and the fact that it was still under warranty. Armed with this information, Sears was able to schedule a repair person on the spot and give me another reason to buy my next appliance from Sears. And I never even sent in a warranty card!

In the computer industry, we're not nearly as good at obtaining the names of our customers. We can, however, learn from companies in other industries, and apply this knowledge to our own businesses. There are two basic aspects of your specific business that determine how much effort it takes to get customer names.

First, the more expensive your product, the easier it is to get a customer's name. Sears knew about the $800 refrigerator I charged there, but had no idea that I also bought a $5 can of paint. Regardless of how customers pay or where they buy your product, they're more likely to register a $599 copy of Adobe Photoshop than a $36 Disney Lion King CD-ROM. The more expensive and complex your product is, the more likely that users are going to register for technical support, upgrades, and the other benefits of your installed base marketing program. (Though I wouldn't recommend raising the price of your $36 game to $599 solely for name-gathering purposes!)

Second, it's going to be easier to track your installed base if you sell your products directly to consumers than if you sell exclusively through retail channels. Unless you require your sales channel to report end-user sales (Microsoft product managers, for example, receive weekly reports on product sales from major retail stores), you will lose sight of your product as it travels from distributor to retailer to consumer.

But no matter what price range or channel you sell into, the most important aspect of any installed base marketing plan is your ability to locate your customers. You have to get their names! Any direct contact you have with a customer—whether through support calls, e-mail, complaint letters, or, of course, registration cards—should be considered an opportunity to capture customer names and addresses.

Registration Cards: Bad News, Good News

The bad news about product registration cards is that very few customers send them in—especially for low-end products.

What's the good news? If you can get anywhere near 50 percent of your product registration cards returned, you'll be doing better than most of your computer industry peers. There are many ways to increase your return rate. It just requires a different mindset. Time and effort spent marketing your product registration card will be rewarded by many years of successful installed base marketing.

Attack the registration card problem on three fronts. First, make it obvious that you want the card filled out. If you use a "real" card, make sure that it's impossible to overlook when the customer opens the box. Better yet, build the registration process into your software product, including a modem call to a toll-free registration number. Just be sure to give your unnetworked users a paper option.

Second, make it exceedingly simple for customers to provide you with information. Design your registration fields so that they're easy to understand and inviting to fill out. At a minimum, collect your customer's name, mailing address, and e-mail address. Preprint a customer's product and serial number on the card so you'll be sure to capture this information. Keep it short, because a long form will reduce your yields. And don't make them buy stamps; use a prepaid mailer.

Finally, no matter how easy you make the registration process, you're still asking people to spend valuable time doing it. What are you going to give them in return? Here are a few incentives that

Inside This Section

other developers have used with success:

• free technical support for a year (a $72 value)

• a subscription to *Macworld* magazine (a $24 value)

• a quarterly newsletter with tips on how to use your product

• a free item, such as a mouse-pad (please, no more company logos!)

• discounts on other products you sell

• cost reductions on future releases and upgrades

• an opportunity to beta-test new products

• placement on your e-mail distribution list

• a request to be on your advisory council

• a chance to win a free product or trip

While this is by no means an exhaustive list of registration card incentives, it gives you some ideas to experiment with. You're giving customers a reason to register beyond the prospect of receiving "junk mail" from yet another vendor. Create a compelling program, tell your customers about it, and the registration cards will pour in. (For more ideas on building your customer list, see the box, "Seven Ways to Increase Registration Card Returns," compiled by Ivan Levison, a writer who specializes in technology direct mail.)

Building Long-Term Customer Relationships

Citibank, one of the finest marketing companies around, thinks about customers in terms of the value of their lifetime financial transactions. Cellular phone companies also view their customer relationships this way: Phones are practically free (I even saw an ad in which a cellular company offered to pay me to take one!), because the value of a customer is in the long-term service revenue, not the one-time hardware purchase.

Increasingly, the most successful software companies are taking a similar approach, viewing each customer as a lifetime source of revenues. These companies are working hard to build and maintain long-term relationships with customers. Here are a few tactics that can help you foster a sense of loyalty among your customers and bring in more future sales:

• *Publish a newsletter.* It doesn't have to be elaborate. Even an 8.5-by-11-inch quarterly

Seven Ways to Increase Registration Card Returns

By Ivan Levison

When most software publishers start planning their upgrade mailing programs, they worry about things such as setting the right price-point and finding hot bonus offers. But they often don't ask themselves the most important question of all: "How effectively am I obtaining the names of the people who buy my software?"

If you're using an in-box registration card to capture vital information (as opposed to electronic "forced" registration), here are some techniques you can use to start improving return rates immediately.

1. Make your registration card BIG. When your customers open your box, you know they're headed toward your installation disk or CD. This means you have to stop them cold to make them fill in your registration card. Since the goal is to get noticed, why stick to a pathetic, invisible little 7-by-4-inch card? It makes much more sense to start out with an 8.5-by-11-inch sheet that folds down to 5.5 by 8.5 inches. This format gives you a cover surface you can use for some terrific copy that gets the user's attention.

2. Never stick the registration card inside your manual. The first thing you want the user to do is fill in the registration card and drop it in the mail. So why hide it? All too many publishers mix their registration cards in among a pile of fliers and notices, or insert it in the manual where it may never get noticed.

3. Consider putting your registration card in an envelope along with your floppy disks. That's a sure way to get your registration card noticed. That's what Intuit does on the back of the envelope that contains Intuit's Quicken disks. The envelope reads as follows:

Fill out and mail Quicken registration card.

Remove and install program disks.

Take advantage of valuable offers.

4. Provide a clear reason for the user to register. Here are some proven motivators for you to consider:

• free premiums such as fonts, clip-art, charts, templates, books, newsletters, and so on

• free technical support

• special offers on future products (that is, additional sources of revenue for your company!)

• early-bird upgrade special deals

• new product information (add-ins, enhancements, and so on)

• free replacements for damaged disks

5. Don't ask for too much information. Obviously, registration card information is precious, but if you load up the card with too many questions, you'll turn the user off and get nothing. It's much better to ask a few important questions and avoid overloading users.

6. Put a reminder sticker on your first installation disk. Since you know for sure that every single user will pick up your installation disks, why not attach a sticker that says something like this:

Important! Fill in and mail the enclosed registration card while you are still eligible for FREE customer support.

This simple technique can have a significant impact on registration card returns.

7. Invest in your registration card. For all too many publishers, the lowly registration card is a mere afterthought. What a pity! A well-written, expertly designed registration card can have an enormous effect on return rates, as forward-looking publishers have already discovered. Remember, a modest investment in your registration card can result in truly giant increases in revenues.

Ivan Levison is an independent copywriter who writes profit-building direct mail, advertising, brochures, package copy, and other materials for technology companies. For more direct mail tips, check out his Web site at http://www.levison.com or give him a call at 415-461-0672.

self-mailer will keep your company name and product information in front of your customers.

• *Organize customer events.* Hold an open house at your office for local customers. Leverage your trade show efforts by organizing a "customer appreciation" reception at these events.

• *Create a "preferred customer" support line.* Reward your registered users with enhanced technical support features—a shorter wait, escalation to your engineering group, or a lower price.

• *Establish a customer advisory council.* Let registered users apply to become members of an elite customer panel that provides advice on future product plans.

• *Survey your installed base regularly.* At least once a year, conduct a comprehensive survey of your registered users, asking them about product usage, their favorite and most disliked features, and any competitive issues that are important to them.

• *Use the information superhighway.* It's more than a buzzword: Online communication is the most cost-effective way to maintain your relationship with customers. Create a Web page so customers can check out your latest product information and demos. Use a list server to capture prospect names and to quickly notify interested users of upgrades, special offers, and new products.

• *Offer registered users discounts.* Financially reward these users for their loyalty, and they may return the favor by buying other products in your line.

• *Be creative!* Hold a brainstorming session with your company's product development, support, marketing, and production people. Invent new ways to strengthen your relationship with the customer.

For more ideas on building customer loyalty, take a look at the any of the publications listed in the "Installed Base Marketing Resources" box on this page.

Someone Else's Installed Base?

If you're launching your first product or looking for ways to extend your marketing reach, you may be able to tap into another vendor's installed base to fill out your own customer list.

For example, you can easily gain access to several lists of Apple Computer, Inc., customers. (See the "Installed Base Marketing Resources" box for contact information.) One of the best databases of potential customers is available through the Apple Authorized User Group program. Over the last 13 years, Apple has supported the formation and development of a strong user group community. Today, there are nearly 2,300 Apple user groups in the United States and Canada, and hundreds more in other parts of the world. Combined membership in the United States alone is more than 600,000. The word-of-mouth influence that user group members have on product sales makes these customers especially valuable. The typical member earns an average of almost $60,000 per year and influences around $30,000 in annual PC-related purchases.

Working with user groups is an especially cost-effective approach to tapping into Apple's installed base, since most mailings are subsidized by Apple Computer. For example, mailing a data sheet to the leader of each user group costs about the same as a first-class postage stamp.

If your products are sold into large institutions, you may want to consider co-marketing opportunities with the Apple Support Coordinator (ASC) program. Apple Support Coordinators are the key individuals that Apple sales representatives call upon in higher education, government, and corporate institutions. There are currently about 10,000 identified ASCs in corporations, and another 5,000 in colleges and universities. Every quarter, Apple mails product and marketing program information to 15,000 key institutional customers. Like the user group program, developers can participate in this ASC mailing for a nominal price.

Apple frequently runs co-marketing campaigns that let you tap into their installed base, though most tend to be short-term promotions. The trick is to stay in touch with the marketing folks in the consumer, business and government, and education groups to learn about these opportunities in advance.

Apple isn't the only company to provide access to their installed base. About a month ago, I received a catalog from Intuit. It included information on new Intuit products, but it also included advertisements for several unrelated products available from other companies. "Bundling" your product information with another company's mailing is another cost-effective way to reach new potential customers.

The Real Payback

You've got the names. Your customers feel like you care about them, since you've worked hard to develop a relationship with them. Now you just have to sell them something. After all, it's revenue that makes this all worthwhile. The best way to be sure that you're making the right offers to your installed base is to check with some of your customers. If you have an advisory council, try out your ideas on them before you roll them out to your entire customer base.

The simplest sale is the upgrade. The hardest part about selling upgrades is pricing. How much is it worth to go from release 3.12 to 3.13? How about from 3.13 to 4.0? And should it cost more for customers who skipped a release? Only your customers can help you with the answers.

Installed Base Marketing Resources

Here's a short list of publications and contacts that can provide you with a wealth of ideas on marketing to your installed base of customers.

• *The Direct Marketing Handbook* by Edward L. Nash (McGraw-Hill, 1992).

• *Database Marketing* by Edward L. Nash (McGraw-Hill, 1993).

• *How to Drive Your Competition Crazy* by Guy Kawasaki (Hyperion, 1995).

• *Customers for Life* by Carl Sewell (Doubleday, 1990).

• *Raving Fans* by Ken Blanchard and Sheldon Bowles (Morrow, 1993).

• *Marketing With Computer User Groups* by Sam Decker (User Group Connection, 1995).

• "The Levison Letter—Action Ideas for Better Marketing Communications," 415-461-7738.

• Apple Authorized User Group Program and Apple Support Coordinator Program mailings are managed by the User Group Connection. For mailing service information, contact this organization at 408-461-5700, send e-mail to info@ugconnection.org, or check out their Web page at the location http://www.ugconnection.org.

Another potential source of incremental revenue from your installed base is sales of "companion" products. A companion product can be a market-specific template or set of macros (say, a Photoshop filter or a Quark prepress script) that enhances the functionality of your main application. Another type of companion product is one that performs complementary functions for the same type of customer—for example, a calendar program sold to the installed base of a contact management package.

Companion products may be marketed as joint ventures with other developers where you "swap" customers, or they may be related products from your own company. And if you can't find a third-party companion product, consider building one from scratch.

Another great way to sell incremental products is by organizing a club or interest group around your customers' interests. At the software distribution level, KidSoft has done an excellent job of installed base selling through their *Club KidSoft Magazine* and demo CD-ROM. Club KidSoft materials entertain computer-using kids, all the while building customer loyalty and "soft-selling" product offerings.

Leverage, Leverage, Leverage

Leverage may be the most overused term in all of marketing, but your installed base may be the greatest leverage you have. Remember, it costs up to five times as much to get a new customer as it does to sell to a current customer. And the increased satisfaction you garner from a well-designed installed base campaign can turn your current users into highly motivated evangelists for your products. ♣

Ray Kaupp (rkaupp@ugconnection.org) is the president of User Group Connection, an organization that helps Macintosh and Windows developers market to user groups around the world.

Business Feature

Producing "Bug-Less" Software—Part 1: The Testing Process

By Victor J. Hnyp, President, Prosoft Labs

Question: Which of the following types of software are most likely to be bug-free?

(a) freeware
(b) shareware
(c) commercial applications
(d) commercial utilities

Answer: None of the above.

Witness the first rule of software testing—No application, utility, or any other software package is completely devoid of bugs. You can test a software product for years, but not find all of them.

Since 1985, Prosoft Labs has helped companies identify and eradicate bugs. We've tested commercial software for publishers of Mac OS and Windows-based applications, cross-platform utilities, and multimedia products. And through these efforts, we've developed an effective methodology for producing software with the least possible number of bugs—in other words, "bug-less" software.

This two-part article provides you with information that will help you create more reliable software. Part 1 discusses basic strategies and processes essential to software testing. Part 2 describes some of our tried-and-true bug-chasing tactics. If you're new to the testing process, this first article will give you a better understanding of what test engineers do, and it may even help you become a better tester of your own products. If you're already an expert software tester, you'll probably find the technical tips in Part 2 to be most useful.

Bugs: An Industry Infestation

From Internet access programs to learning games to business applications, buyers are confronted with flashy packaging, glitzy promotions, and "bargain" software. At the same time, dozens of new computers, CD-ROM drives, printers, and other peripherals are being introduced. Operating systems are enhanced at least once a year. Drivers, screen savers, and utilities abound. And many of these products are released to consumers with very little (or the wrong) testing.

Every day, hundreds of systems are purchased by first-time buyers. These novices are smitten by promises of "ease of use" and "plug and play." They unpack their new products, and sometimes within minutes they're faced with system errors, debug messages, and frozen computer systems. Witness the birth of unhappy customers.

How can you be sure *your* software product doesn't contribute to a customer's unhappiness? Since software can't be 100 percent bug-free, we as developers should concentrate on producing products with the fewest bugs possible. Hence the term *bug-less* software.

Why Can't Software Be Bug-Free?

It's not just your imagination: It *is* getting harder to find and eradicate software bugs. There are several reasons behind this trend:

• *Configuration proliferation.* Back in 1984, there was one model of Macintosh computer, and Apple Computer, Inc., provided users with most peripherals and drivers. If your program ran fine on the Macintosh on your desk, chances are that it ran fine on any Macintosh of that era. Today, there are dozens of Macintosh models that can be customized and configured a thousand different ways—all of which makes it more difficult to thoroughly test new products.

• *Multiple standards.* As the industry migrates toward cross-platform applications, programmers must support multiple standards. Programs must address data format differences, and byte-swapping situations such as big-endian versus little-endian. Windows-based computers support different sound, color table, and movie formats than Macintosh computers. Driver software must contend with NuBus™ and PCI bus structures.

• *Program size.* Today's software has grown in size in order to support extensive computing options. In 1984, programs 64K in size were considered large, whereas today a word processor can require 6 megabytes or more! Large programs have complex decision paths and often require virtual memory, which prevents you from knowing when pieces of your code will get swapped to disk.

• *Programming complexity.* As the power of the Mac OS platform grows, so does the complexity of its programming environment. Macintosh programmers must now support CISC processors from the 68000 family, RISC processors from the PowerPC family, and numerous operating systems from old (6.0x) to new (7.5.2). We must contend with and support technologies such as Apple events, OpenDoc, AOCE, and QuickDraw GX, and the list goes on! Cooperative multitasking is more important than ever. Drivers must coexist with other drivers. Applications must support multiple drivers and coexist with other applications.

The programmer's job is getting tougher, and it's easier to make mistakes. But by intelligently planning your software quality assurance (SQA) testing process, you can maximize your testing resources, while minimizing the number of bugs in your new product.

Building "Bomb Shelters"

It would take several hundred years to fully test a software package. There are virtually billions of combinations of variables to test in any given piece of software. And it often seems that the more time you have to test a program, the more tests you'll realize you have to perform.

This is why code path analysis (CPA) utilities have enjoyed increased popularity among SQA teams. CPA utilities analyze a program and determine the many possible decision paths that program flow can take. The utility then creates scripts that repeatedly run under automated test environments and change values of variables. The intent is to make sure that *each line* of code is exercised within an application. This works well on the surface, but it can lead to a false sense of security. Automated testing doesn't approximate application usage under real-world conditions.

The real world is filled with users who don't follow the rules. Users can (and do) attempt strange things that automated testing and code path analysis scripts can *never* reproduce.

The real world is filled with users who don't follow the rules. Users can (and do) attempt strange things that automated testing and code path analysis scripts can *never* reproduce. Users try every imaginable combination of keystrokes. They change monitor depth for no apparent reason. They switch in and out of applications on a whim. They may shut a computer off by pulling the plug from the wall.

No matter how hard you try, you can never completely test a software product in the myriad of ways a customer will use it. As developers, we can only anticipate the fallout, then build a better "bomb shelter."

In essence, every testing schedule is an accelerated testing schedule! From what we've already learned, it's apparent that it takes months (and possibly years) of testing to try every program function in every combination of machine, operating system, monitor, and printer.

The trick to successful software testing is not to test *all* possible combinations, but to pick a cross-section of tests that are most likely to produce anomalies. Use testing tools wherever possible, but don't rely too heavily on automated tests. Concentrate on tests that quickly pinpoint problems that may never surface through automated means.

Pre-Alpha Stage—Generating the Test Plan

It helps to start testing as early as possible in a program's development cycle. The later a problem is discovered in the development cycle, the more time, money, and effort it takes to fix it. Many project managers make the mistake of testing a product only after it has reached the late alpha or early beta stage. Then it's too late in the game to make meaningful revisions. A better approach is to have your software quality assurance (SQA) team involved from the very beginning, when the "requirements" document is distributed. This gives your test lead engineer a head start in preparing a detailed test plan.

The SQA test plan is your test team's road map of the work ahead of them. It helps your company discover, identify, report, and track problems in a software product. By performing this service, SQA teams improve the quality and reliability of a program. This in turn results in fewer technical support calls, fewer bug-fix updates, and better margins for software publishers.

With a good SQA plan, project managers can sleep at night knowing that most bugs (and hopefully all the serious ones) are eradicated from the shipping product. A poor SQA plan, or none at all, puts the product (and the project manager) at risk of failing in the marketplace. Releasing buggy software can result in a tarnished company image and possible lawsuits.

It's the test lead engineer's job to determine which tools need to be purchased and which should be written from scratch. The test lead designs test cases that are used to test functionality, compatibility, and anticipated problem areas. The test lead designs a test matrix that will serve as a check-off sheet for tests as they are completed.

From the requirements document, a development team can identify areas of the program that must pass "race" conditions—conditions in which timing or speed of execution is critical. These pieces should be coded and tested first, so that if problems are found, another approach can be engineered. This is extremely important for code that is interrupt-driven.

Test teams can establish benchmarks based on execution speed, accuracy, or any other measurable criteria. These benchmarks become test cases that are checked with each new build of the program to make sure things aren't slipping beyond a given threshold. The test team can use benchmarks to test a product against a competitor's, then provide valuable feedback to the development team.

By involving a test lead engineer early in the process, the project manager can get useful feedback on the testability of the product. Through the test lead's guidance, the requirements document can stipulate compile switch settings that allow greater feedback to test engineers about the internal state of the program. Prior to final build, these compile switches can be turned off so that the program's final delivery size or execution speed are not affected.

Alpha Stage

The alpha stage is when "real" testing begins. Alpha testing is performed through a *test cycle*—a full test of all applicable test cases that were designed for the given deliverable, along with updates to the bug-tracking database. At the close of a test cycle, the test lead produces a summary report of the bugs found, plus a summary report of all known problems. Test cycles repeat through alpha, beta, and final candidate stages until the product ships. The first test cycles take the longest amount of time (sometimes lasting a few weeks). As development progresses, test cycles tend to get shorter, because (hopefully) less of the program needs to be retested each time.

Usually, a test cycle consists of the following steps (shown in order):

• *"QuickLook" acceptance.* One or two test team members receive a build from the development team. They spend a short amount of time (15 minutes or so) making sure the application launches and provides basic functionality. If menus don't work, or if the program crashes or runs out of memory with no provocation, then there's no use in testing this build any further. If the program fails quickLook acceptance, it's returned to the development team.

• *Deliverable acceptance.* The program is checked for functionality against the promised deliverable. The deliverable states that a certain dialog box or other piece of code should be implemented. If the tester can't navigate to that piece of code (a menu is missing or disabled, the dialog box won't appear, and so on), then that fact is entered as a problem in the bug-tracking database. If too many items are missing, the build might be rejected completely.

• *Regression testing.* Regression testing is testing that is performed every time a bug is claimed to be fixed. The test that produced the original bug is repeated, and the results are entered into the bug-tracking database. If the bug has indeed been fixed, the bug is "closed" in the database. If the bug is not fixed, an explanation for why the bug wasn't fixed correctly is entered into the database, and the bug is returned to the development team for another try.

A second part of regression testing entails making sure that a bug fix doesn't break something else. This is extremely important when a new decision path is added to an already complicated section of the program. All related decisions in the decision path should be regression-tested in order to make sure they still function according to specification.

Sometimes bugs mask other bugs. This means that once the first bug is fixed, the second bug becomes apparent. It's the nature of software engineers to concentrate on fixing documented bugs, then moving on to the next bug in the bug database. It is the tester's responsibility (through regression testing) to find any bugs that become "unmasked" by any given bug fix.

• *Test-case execution.* With each pass through the test cycle, more of the development has been completed. More of the test plan is executed and more of the test matrix is checked off. Test-case execution includes all types of testing, including compatibility, configuration, and stress. (Some of these specific test cases will be discussed in detail in the second part of this article, which will run next month.)

During test-case execution, it's important that testers stick to the testing designated for each deliverable. Deviation from the test plan can lead to testing parts of the program that have not been officially released by the development team, resulting in bug reports that shouldn't be entered into the database. The bugs show up in summary reports, graphs, and charts, and throw off the true state of the development effort. This tends to cause unnecessary frustration for the development team!

• *Ad-hoc testing,* also known as *mainstream usage testing,* doesn't demand that testers follow a predefined plan, but leaves them free to use the program in a way customers would use it in real-world situations. This often uncovers bugs that may elude a strict, regimented test plan.

Better testers produce good results through ad-hoc testing because it gives them a chance to exercise a program in the areas where they feel it's weakest. Each tester should be encouraged to do some amount of ad-hoc testing during each test cycle.

Beta Stage

The end is in sight! At the beta stage, the program and documentation should be functionally complete, and all known bugs should be accounted for (either fixed, in progress, or deferred). The cycle of testing continues through each beta build of the program, but the test team is not held back from testing any pieces.

Normally, project managers enlist the help of end-user testers during the beta stage. These beta testers should be far removed from the planning or development of the project, so that they don't feel emotionally tied to the end product. Beta testers should be warned that they should expect to find crash bugs and other problems that may corrupt their data.

Outside beta testing usually falls under the jurisdiction of the test lead engineer. The test lead is the liaison between the outside testers and the SQA team members. All bugs reported by beta testers should be regression-tested by the test team before the bugs are committed to the bug database.

Final Candidate

After a period of repeated test cycles during beta testing, there comes a time where the outside beta testers and the in-house SQA team stop finding bugs. Remember, this doesn't mean there aren't any bugs left! What this does mean is that the remaining bugs aren't serious enough or prevalent enough to surface in the real world. Congratulations! You're at the final candidate stage.

Prior to announcing a final candidate build, the test team should review and regression-check all bugs (open, closed, deferred, and so on) that were entered into the bug-tracking database. Once the review is complete, the test lead engineer compiles a final report. All deferred or unresolved bugs should be addressed in an executive summary, with special emphasis placed on bugs that are slated to be fixed in future versions of the program.

The final report should be distributed to all parties that were involved in the project. It's especially important that the technical support team reads and understands the final report, since the report will spell out any problems customers may encounter with this version of the program.

Golden Master

In most companies, the test lead engineer is responsible for producing golden master disks or a CD-ROM. This final master disk should have all the packaging, labels, and documentation of a final off-the-shelf released product.

The golden master disks should be produced on a system that has been checked for integrity. Integrity means that nothing but the necessary Mac OS software is in the System Folder. No custom icons, games, or any other software should exist on this machine. The computer's hard disk should be checked for integrity (correctness of the directory bitmap, and so on) and scanned for viruses. All initial window positions should be double-checked to make sure they open in the correct location on screen when folders and icons are double-clicked.

Once the first few floppy disks or CD-ROM discs come back from duplication, the test lead engineer should check them for integrity—again! Although it's not common, disk duplication can introduce new problems, including invasion of viruses.

On the Importance of Process

Software testing is the process of discovering, identifying, reporting and tracking problems in a product. The Software Quality Assurance team performs various tests such as compatibility, configuration, and stress, through a cycle of testing that starts at alpha stage and goes through final candidate. There are special tests that are performed at the golden master stage to ensure integrity of the product before and after it goes through the duplication process. Having an organized software testing process in place, which everyone understands, is the foundation to creating a bug-less software product. ♣

Victor Hnyp (victor@prosoftlabs.com) is president of Prosoft Labs, an engineering firm based in Pleasanton, California (510-426-6100) that specializes in the testing of Mac OS– and Windows-based hardware and software.

Editor's note: Look for the second half of this article, "Producing 'Bug-Less' Software—Part 2: Bug Chasing Tactics," in next month's issue.

Listings

Developer University Schedule

Developer University (DU) offers a broad range of Mac OS and Newton programming instruction through hands-on classes and self-paced training products. Classes are offered in Cupertino, California, and through selected third-party trainers.

The following is a list of upcoming DU course offerings, including when and where they're offered and how much they cost.

Advanced C++/5 days/$1,000

March 4–8	Cupertino, CA

Apple Events/AppleScript Programming
5 days/$1,500

January 29 –February 2	Portsmouth, NH
February 12–16	Cupertino, CA

Creating Apple Guide Help Systems
4 days/$1,200

February 12–15	Cupertino, CA

Creating OpenDoc Parts/5 days/$1,500

January 15–19	Cupertino, CA
February 19–23	Cupertino, CA
March 18–22	Cupertino, CA

Macintosh Debugging: Strategies & Techniques
3 days/$900

January 22–24	Cupertino, CA
March 13–15	Portsmouth, NH

Multimedia Development with QuickTime VR
3 days/$900

January 16–18	Cupertino, CA
February 20–22	Cupertino, CA
March 19–21	Cupertino, CA

Newton Programming: Essentials
5 days/$1,500

January 15–19	Cupertino, CA
February 12–16	Cupertino, CA
March 11–15	Cupertino, CA

Newton Programming: OS Enhancements
5 days/$1,500

January 29–February 2	Cupertino, CA
February 26–March 1	Cupertino, CA

Programming with MacApp
On demand—call DU Registrar for more information

Programming with QuickDraw 3D
3 days/$900

January 15–17	Cupertino, CA

Programming with QuickDraw GX/4 days
On demand—call DU Registrar for more information

QuickStart Mac OS Programming/5 days/$1,500

January 29–February 2	Cupertino, CA
March 4–8	Cupertino, CA

Scripting with AppleScript/2 days/$600

January 22–23	Cupertino, CA
February 26–27	Cupertino, CA

Writing Reusable Code
3 days/$900

February 5–7	Cupertino, CA

To register for a class or to get a complete course description by fax, call the Developer University Registrar at 408-974-4897.

Course descriptions can also be found electronically at the following locations:

• **AppleLink:** Developer Support:Developer Services:Apple Information Resources:Developer Training:Developer University

• **eWorld:** Computer Center:Apple Customer Center:Apple Developer Services:Developer Information:Developer University

• **Internet:** http://dev.info.apple.com/du.html

• **America Online:** Computing:Computing Forums:Development:Mac Development Q&A:Developer University ♣

The Internet Page

This feature is devoted to informing you about where you can go on the Internet for online information about Apple Computer, Inc.; its products, technologies, and programs; Mac OS and Newton programming; and other subjects that pertain to the business of computer product development. You'll find this feature particularly helpful when you view it at the *Apple Directions* Web page (located at http://dev.info.apple.com/). There, all the names of the locations listed in this article are linked to the sites themselves; clicking the names will take you directly to the relevant Internet locations. We'll update this feature every month, based both on what Apple is doing on the Internet and on your feedback.

Apple Sites

This section describes World Wide Web sites maintained by Apple Computer.

http://dev.info.apple.com/
This site contains the Apple Developer Services and Products page, and is probably the most important World Wide Web page for you. It contains Apple Directions Express with live links to other Internet locations and the online versions of *Apple Directions,* and *develop,* the Apple Technical Journal. It also links you to a variety of other sites that give you access to the gamut of Apple's online developer support services.

http://www.apple.com/
This site contains the Apple Computer home page, with links that will let you go to just about all the other Internet sites maintained by Apple, even the ones listed separately here.

http://www.info.apple.com/macos/
This is the Mac OS Web site. You can go here for the latest information on the Mac OS, including details about Copland, white papers on new Mac OS technologies, marketing and strategic information, and other items to help you develop new Mac OS products.

http://dev.info.apple.com/dev/technotes/Main.html
This is the Web site for the Apple technical notes series. Check it out this month for the new technical notes, as well as for recently posted guidelines in case you want to contribute your own technical notes.

http://www.info.apple.com/pacific/
This is the Apple Pacific home page, with information about Apple offices and developer support in the Pacific region, including Japan, Australia, Canada, and Latin America.

http://www.euro.apple.com/
This is the front door for information about Apple activities—including developer services—in Europe, with pointers to Internet sites for specific countries. Sites are currently established for Apple Norway, Apple Italy, Apple Germany, and Apple Benelux.

www.info.apple.com/newton
www.info.apple.com/dev/newton
These are Apple's Newton Web pages; the first is called the World of Newton, and it houses a variety of information about the Newton platform, primarily for customers. The second is the Newton developer site.

http://coretools.apple.com/opendoc
This is the site of Apple's OpenDoc home page, featuring Developer Depot, where you can find the latest OpenDoc release, documentation, and tools, and Developer Showcase, from which you can download and sample actual OpenDoc parts!

http://www.info.apple.com/dev/thirdparty/
Apple Fellow Guy Kawasaki set up this Web page to list your hardware and software products. Fill out the form located at the site to add your products; that way, everybody on the 'net can find out about what you're up to.

http://www.amp.apple.com
This is the site of the Apple Multimedia Program (AMP) home page. If you're a multimedia developer or considering getting into multimedia, you'll want to check out the information on this page about Apple's multimedia technologies, as well as the links provided to other Internet sources about multimedia. It also includes the AMP Member Showcase, a searchable database of multimedia developers.

http://www.apple.com/whymac/
The official source for official Apple ammunition to fight the war against Windows 95, including the extensive series of Windows 95 vs. Macintosh Updates, prepared in the wake of the Windows 95 release.

http://www.info.apple.com/gomobile/
This site contains complete information about PowerBook computers and the full line of Apple mobile computing solutions.

http://www.info.apple.com/dev/evangelism/powertalk/
Apple's PowerTalk home page, with resources for PowerTalk programmers. It currently contains the StarNine gateways recently licensed by Apple: Mail*Link Internet for PowerTalk, Mail*Link MS for PowerTalk, and Mail*Link QM for PowerTalk, which give Mac OS users access to Internet mail, StarNine Mail, or C.E. Software's QuickMail. You can download the gateways for no charge.

http://www.info.apple.com/qd3d/
Apple's QuickDraw 3D home page contains everything you need to know about QuickDraw 3D, including QuickDraw 3D applications you can "test drive."

http://www.info.apple.com/powermac/powermac.html
http://www.info.apple.com/ppc/ppchome.html
These are two useful sites for information about Power Macintosh computers.

http://quicktime.apple.com
This site contains the QuickTime Continuum page with news and technical and marketing information about QuickTime.

http://qtvr.quicktime.apple.com
This is the location of the QuickTime VR page.

http://www.info.apple.com/gx/gx.html
This site contains the QuickDraw GX home page.

http://www.info.apple.com/education
Here's where you'll find the Apple Education home page with information about Macintosh computers for the education markets. You can also use online forms located at this site to request product specifications, information about the Apple Education Series (bundled products), and technical support from Apple engineers.

http://www.mae.apple.com
The Macintosh Application Environment (MAE) home page.

http://pippin.apple.com
The Pippin Web page contains the latest information about Apple's PowerPC processor–based, low-cost CD playback device.

http://www.eworld.com/
Go to this location to find content and services from eWorld, Apple's online service.

http://www.apple.com/documents/otherappleservers.html
This is the site of the Apple Internet Servers page. Once you've exhausted the obvious Web sites just listed, this page will give you ideas about where else to go on the Internet to find the information you need. This page includes lists of other Web sites as well as Gopher and FTP sites.

Non-Apple Sites

We can't guarantee the information the following sites contain, since they're not created by Apple, but we think you'll find them useful and interesting. They're listed alphabetically, by site address.

ftp://ftp.sri.ucl.ac.be/pub/
A Belgian reader alerted us to this FTP site, where you can find French versions of Macintosh Internet software, including Eudora, Fetch, Finger, FTPd, Gopher Surfer, NCSA Mosaic, NCSA Telnet, TurboGopher, and many other applications. An affiliated World Wide Web site (http://www.sri.ucl .ac.be/SRI/jpk/logIntMacFr.html) describes—*en Français*—what's available at the FTP site.

http://home.mcom.com/home/internet-search.html
This site contains the Internet Search page, which gives you access to InfoSeek, Lycos, and WebCrawler, three excellent Web search engines. If you use Netscape, you can reach this location just by clicking the Net Search button.

http://hyperarchive.lcs.mit.edu/HyperArchive/Abstracts/snd/util/ HyperArchive.html
This site, which is maintained by some of the good folks at the Massachusetts Institute of Technology, contains 200 Macintosh applications and utilities (give or take a few) that can be downloaded for the price of connection charges.

http://rever.nmsu.edu/elharo/faq/vendor.html
The Macintosh Vendor Directory, a directory of companies that make and sell products for the Macintosh computer.

http://www.ape.com/webstar/
This site provides a database of all the Macintosh computer-based Web sites its owner can find, so far nearly 1,000 entries strong. It also lists outstanding Macintosh sites, as well as some losers that refuse to move to a Macintosh server solution.

http://www.astro.nwu.edu/lentz/mac/programming/tools.html
This site is a terrific source for Apple and non-Apple Macintosh programming tools.

http://www.cilabs.org/
The location of the CI Labs home page, which contains a great deal of OpenDoc content.

http://www.class.com/MacTech/URLs.html
This site contains a useful list from *MacTech* magazine of Internet locations on a variety of subjects, most of them pertaining to the technical aspects of Mac OS development.

http://www.cs.brandeis.edu/~xray/mac.html
Nathan's Everything Macintosh page is a treasure trove of Macintosh information; it contains a thorough listing of Apple and other corporate sites that pertain to Mac OS development as well as games, e-mail mailing lists, periodicals, a listing of FTP sites, software archives, and even Apple II information.

http://www.digitool.com/
If you're looking for information on Macintosh Common Lisp (MCL), this is the place to go. Digitool's Web page contains information on MCL 3.0 and other MCL products, as well.

http://www.freepress.com/myee/ultimate_mac.html
This site contains the ULTIMATE Macintosh page, including more Mac OS information and software than you could possibly imagine exists. We think you'll particularly enjoy the software archives and games sites, from which you can download real-live software and play with it.

http://www.guideworks.com/
This non-Apple site is the location of the guideWorks home page; it contains so much information about Apple Guide that you can think of it as the Apple Guide home-away-from-home page.

http://www.icsi.net/~crfrank/newpcTales2.toc.html
Check this page out! An enterprising, very pro-Macintosh NASA employee put it together to debunk common Macintosh myths. There's a ton of good data here to help you do the same.

New This Month/From Our Readers

This list contains Internet "stuff" (for lack of a more descriptive term) we've just become aware of, thanks to *Apple Directions* readers inside and outside Apple. Know of a particularly useful site? Whether it's a Web page, a list server, an FTP site, or a newsgroup, let us know about it and we'll consider adding it to this feature next month, along with your name (!). Send your suggestions to the Internet address a.directions@ applelink.apple.com.

Guy Kawasaki's List Server
Guy calls his list server the EvangeList; it's for official and nonofficial Apple evangelists who want to hear and help spread the good word about Apple. Guy gets information from the press, the Web, Apple internal communications, press releases, and his considerable industry contacts, selects the most useful, interesting—and fun—material, and sends it on to the list. When you join, you can expect to get a couple of dozen notes from Guy each week. More than 10,000 people are already on the list; we think you'll want to be, too. For information on how to join, send an e-mail message (any message will do) to macway-request@abs.apple.com.

Don't Forget About Apple Directions Express!
This is a reminder to subscribe to Apple Directions Express, our online digest of business news and information from Apple, sent to you biweekly over the Internet. It includes pointers to Internet locations and other sources of more detailed information. It's free, it's up-to-the-minute, and it's a terrific source for information you need about Apple Computer (even if we do say so ourselves!).

Here's what reader Alex Gollner said about issue #2:

It has just the right balance of structure-summary-content. All the stories are relevant to me. . . . Each story tells me the right amount of information, and gives me the links should I want more.

You can subscribe by sending e-mail to adirections@thing1.info.apple .com. In the subject field for your message, type the string "subscribe <your real name>".

http://www.opendoc.apple.com
You can download the just-released OpenDoc software development kit from this official Apple site.

http://www.ugconnection.org/vendors/vendors.html
This is the Web site for the User Group Connection. It contains resources and services to market your products to Apple's most influential and enthusiastic users: user groups. The site includes an online user group locator, a listing of user group Web pages, a monthly newsletter, and excerpts from a new book called *How to Market With Computer User Groups.*

http://www.islandnet.com/~quill/c3data.html
This is a non-Apple site containing The Complete Conflict Compendium; its owners, Quill Services Ltd. and MacSymum POWER Systems, have the goal of listing all software conflicts on the Macintosh computer and the cures for them. Forms at the site let you report conflicts and register your e-mail address so you can be notified if others report conflicts having to do with your products.

http://www.machack.com/
This is the home page for MacHack, an annual conference for hackers that work on the Macintosh computer. The conference features an all-night hack contest, usually with very interesting results. At the 1995 conference, attendees decided on the ten issues they'd most like Apple Computer to address. You can find Apple's responses to those issues, results from the hack contest, and information about the next conference at this Web site.

http://www.memphisweb.com/mathew/default.html
http://www.memphisweb.com/nammac/default.html
Need to find programmers and others to work on developing Macintosh products? Go to these locations for help. The first is called *MATHEW,* which stands for *Macintosh Talent, Help Wanted.* All companies and job seekers can post want-ads there for free, and the database is searchable by city, state, and expertise. The second site is a free directory of individuals and companies available for short-term tasks or contract work, also searchable by city, state, and expertise.

http://www.utu.fi/~jsirkia/mac/
This is the Cult of Macintosh Web site, another "everything Macintosh" compendium of information for Macintosh lovers. It contains dozens (maybe more than 100) links to other sites containing everything you'd ever want to know about the Macintosh computer, as well as sites with tools, free software, and so on.

http://www.dsu.edu/~bitzm/why_buy_mac/index.html
This site contains a compelling article called "Why Should I Buy an Apple Macintosh System?"

http://www.umd.umich.edu/~nhughes/dna/stories/adamson95.html
This is Douglas Adams's Web page, containing his thoughts on Windows 95. We think you'll love it.

http://www.kaidan.com
Here's a reader-recommended site that will be especially interesting to QuickTime VR developers. The site contains information about add-on lenses and QuickTime VR camera mounts for QuickTake cameras.

http://www.metrowerks.com/
This is the Metrowerks Web site, with information about its CodeWarrior PowerPC development tool.

http://www.nisus-soft.com/~nisus/
The location of the Nisus Software home page, which we list partly because of its clever layout. The page looks like a Macintosh desktop; clicking the icons on the desktop takes you to Nisus's various Web postings. Just for fun, click the Trash icon and see where you end up! ♣

Errata and Clarifications

Here at *Apple Directions* we want to be as accurate and complete as possible, to give you the best possible information on which to make your business decisions. Because of this, we need to correct a mistake in the article "Copland Drivers—Time to Put the Pedal to the Metal," which appeared on page 17 of the December 1995 issue.

In the section "High-Level Families," the article stated:

You can implement Copland printer drivers using the QuickDraw GX printing interface model. Existing QuickDraw drivers will work without changes under Copland, but at slower speeds and without the many improved performance features of QuickDraw GX.

The second sentence is in error and should read as follows:

Existing QuickDraw GX drivers will work without changes under Copland, but at slower speeds than if they are updated to take full advantage of Open Transport.

The point that needs to be clarified here is that, for architectural reasons, *no QuickDraw printer drivers will work with Copland.* Copland will include a single printing architecture that provides the features of both QuickDraw and QuickDraw GX printer drivers, with the same or better performance that what Mac OS users experience today. In addition, Apple has pledged that *QuickDraw GX printer drivers will continue to work with Copland* (assuming the drivers have no other incompatibilities with Copland).

What does this affect? Not your software—QuickDraw and QuickDraw GX programs will continue to work under the Copland operating system (assuming that other Copland compatibility issues, if any, have been resolved).

Will your customers' printers be affected by the switch to Copland? It depends on the type of printer. Since Apple already has a LaserWriter GX driver (which prints to most if not all PostScript™ printers), anyone with a PostScript printer will still be able to print while running Copland. However, if you sell a printer and ship your own printer driver with it (presumably to access printer-specific features), you will want to prepare for the Copland operating system by creating a QuickDraw GX printer driver.

If you sell a non-PostScript printer, your customers will not be able to print under Copland while using your current printer driver. The solution here is for you to create a QuickDraw GX printer driver that runs with today's System 7.5; such a driver will continue to work with the Copland operating system. ♣

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 U.S. APDA office from abroad, the number is 716-871-6555. You can also reach us by AppleLink at APDA or by e-mail at APDA@applelink.apple.com. More detailed APDA ordering information is available at the following locations:

- Internet: http://www.info.apple.com/dev/apda.html
- AppleLink: Developer Support:Developer Services:APDA
- eWorld: in the Developer Corner of the Computer Center