Inside this issue

TECHNOLOGY & INDUSTRY

MORE THAN 125 NEW QuickTime products were announced or shown at the San Francisco Macworld last month, and 75+ developers participated in the Apple comarketing activities at the conference. A list of companies who participated is featured.

see the News folder
***

NEW PRODUCTS from Apple and developers are available from APDA. The list of APDA top ten sellers is led by DAL and E.T.O. products.

see the News folder
***

THE APPLE PACIFIC Market Forum was held in January in San Francisco. More than 500 developers, distributors, republishers, consultants, and Apple employees participated. Audio tapes of the sessions are available for purchase.

see the News folder
***

CHECK OUT THIS month's CD, "20,000 Leagues Under the CD." It features the February issue of *develop* magazine, and also includes the new release of System 7 Tune-Up, tools demos, the 1992 Tools Advisor, the *Apple Event Registry*, three Developer University demos, and much more.

see the News folder
***

THE CHINESE government has signed a Memorandum of Understanding that may im-prove protection of developers' software in China.

see the News folder
***

TOG RESPONDS to a reader's query and addresses the "paperless office" concept—with insights into authors' rights, font control, relative fonts, midlevel interactivity, and other issues.

see the Tog folder
***

BUSINESS & MARKETING

POSITIONING A PRODUCT properly is a key to marketing success. Regis McKenna's Geoffrey Moore describes one approach, the "elevator test," that can help you concisely define your product's position in the marketplace.

see the Business & Marketing folder
***

THE KEY TO KEEPING customers satisfied is making a company-wide commitment to understanding and serving them. Intuit has several key programs in place that focus on getting the user input needed to keep customers satisfied.

see the Business & Marketing folder

***

APPLE WILL participate in various trade shows during the next few months. For a listing, see GetNextEvent.

see the Business & Marketing folder

***

THE MACINTOSH worldwide installed base topped 6.9 million units in 1991, and is expected to grow to more than 16 million by 1995, according to IDC.

see the Business & Marketing folder

***

APPLE IS PARTICIPATING in the March SPA conference in Seattle, Washington. The company plans to raffle PowerBooks, host an evangelism suite and the Business Center, and more.

see the Business & Marketing folder

***

GET ANSWERS TO MANY of your development questions, expand your knowledge, or find help for your programming projects by tapping into various resources.

see the Business & Marketing folder

IT SHIPPED! is your guide to new and revised third-party products that have recently shipped.

see the Business & Marketing folder

***

Apple Event Registry: Final Version Available

The standards have been defined: The official, golden master edition of the *Apple Event Registry: Standard Suites* has been released by the Apple Events Developer Association (AEDA). This 500-plus-page registry lays the foundation for all Apple event and scripting technology developments in the future. The *Apple Event Registry* is to the Apple Event Object Model as *Inside Macintosh* is to programming the Macintosh. This long-awaited document, now in its final form, defines the standards for Apple events, and Apple is fully committed to supporting it.

Apple events is the standardized messaging language of the Mac- intosh System 7 Interapplication Communication (IAC) technology. The registry defines the interapplication language and documents all registered Apple events, data types, and object classes. If you want to give your customers access to the power of multiapplication scripting, supporting Apple events and the Apple event standards is a requirement. (For a more detailed discussion of Apple events and scripting, see “Scripting for Success,” in the September 1991 issue of *Apple Direct.* An in-depth technical discussion will appear in a future issue of *develop.*)

A preliminary version of the registry was distributed at last year’s Apple Worldwide Developers Conference, but the golden master edition is a significant improvement. The contents—the core suite, in particular—have been streamlined, and it is available as a printed, indexed reference volume (instead of as a HyperCard stack), so it’s easier to use. Also, two new suites—a database suite and a telephony suite—have been added; more suites will be offered as we receive additional input from developers. (We’d like to thank all the Apple event-enthused developers who have given us suggestions for improving the registry and who helped test it.)

The registry is available onthis month’s Developer CD Series Disc (CD path—Dev.CD Feb 92: Tools & Apps(Moof!): OS/Toolbox: Apple Events:AE Registry/Winter ’92), on the E.T.O. Disc #6, and on the AppleLink network (AppleLink path—Developer Support: System 7 Talk:Apple Events Discussion:AE Registry Winter ’92). Printed versions of the registry will soon be available through APDA.

Several support tools and resources are available to help you implement Apple events:

•*Inside Macintosh, The Apple Event Manager chapter.* This updated chapter, a great starting place for those who are new to working with Apple events, is available on the February Developer CD (CD path—Dev. CD Feb 92: Developer Essentials: Technical docs: Inside Macintosh Preview). It provides an introduction to Apple events and Apple-event objects; it also offers examples that illustrate the Object Model, including how to create and specify Apple-event objects and how to resolve them. It describes how to handle the four required Apple events, how to create and send them, and much more.

•*Developer University Course.* Developer University’s upcoming “Advanced System 7” course teaches developers how to use the Apple events Object Model and Open Scripting Architecture to write applications that communicate with each other. The five-day course will debut in late March. For more information, contact the Developer University Registrar at (408) 974-6215 or send an AppleLink message to DEVUNIV.

- *Apple Events AppleLink Forum.* An Apple events discussion is available on the AppleLink System 7 Talk bulletin board. Using this forum, you can share ideas and discuss problems with other developers who are implementing Apple events. It also provides a wide range of tools, sample code, documentation, and other resources, including a prerelease version of the Apple Event Software Development Toolkit. ◆

NEW SOURCE FOR APPLE II TOOLS

APDA has announced that Resource Central, previously known as A2-Central, has been granted the right to distribute all Apple-label Apple II development products. This means that all current Apple II development products previously sold through APDA should now be purchased through Resource Central.

Located in Overland Park, Kansas, Resource Central distributes Apple II-related publications, hardware, and development tools, and manages Apple II areas for on-line services. The company has supported the Apple II development community since 1985 and is widely known as the sponsor of the A2-Central Summer Conference.

With this transition, Apple II developers will be working with a company whose entire business is optimized to sell and support Apple II development tools.

Products such as Apple II Pascal v. 1.3, MPW IIGS, and the *Apple II Reference Manuals* can now be found in Resource Central's product catalog. The company provides technical support for all of its products—including these latest Apple-label additions—via mail, fax, and phone, and on-line through GEnie. It also guarantees compatibility, and promises that if customers are dissatisfied with anything it will work to rectify the situation, even if it means refunding money.

For more information or to place an order, contact Resource Central at P. O. Box 11250, Overland Park, KS 66207; (913) 469-6502. ◆

Inheritance Is the Best Revenge

MacApp 3.0 handles System 7...and more

by Gregg Williams,
Apple Direct Staff

Here's an interesting recipe: Take the source code of a conventional Macintosh application. Find and circle in red the code that implements windows. Now circle the code that does window scrolling. Ditto for window dragging and resizing. Ditto for printing, menus, mouse tracking, and button and check-box manipulation. Ditto for exception handling, memory management, and special-case support for different Macintosh models. Finally, circle all the code that implements System 7: Apple events, the Edition Manager, and Balloon Help. (You do support System 7, don't you?)

Now look at the source code again. A lot of red ink, right? That's a lot of source code. With a conventional application, you have to write (or copy and adapt) all that code, make sure it runs on every model of the Macintosh product line, and hope it runs correctly on future Macintosh models. And you do a lot of this work just to implement standard user-interface behavior that, *if you're lucky,* your users will take for granted. Why do you keep reinventing the wheel?

Why not "inherit" the code from someone who's done it before? Ever since 1986, Apple has offered MacApp, an object-oriented framework and class library that takes care of most of the above details, allowing you to spend the bulk of your time programming the "meat" of your application—that is, the unique content of your vision, the stuff that only you can do. MacApp was a good idea when it came out, and it keeps getting better. Apple engineers have been improving and evolving it for more than six years, and MacApp 3.0 represents the second major revision of the original product.

The occasion for this article is the forthcoming release of MacApp 3.0, a major revision that makes MacApp even more powerful and easy to use than before. One nontrivial new feature is the built-in support for System 7 features that should be in every application—the required Apple events, the Edition Manager, and Balloon Help.

Also, keep in mind that well-behaved MacApp programs are more likely than conventional programs to work on both current and future models of the Macintosh. (And don't forget that using MacApp gives you valuable object-oriented-programming experience—something you'll need if you want to create next-generation software for Taligent's Pink operating system.)

Do you need any more reasons for learning MacApp 3.0?

This article emphasizes the new features of MacApp 3.0. For a brief summary of what MacApp is, see the sidebar "(Slightly) Inside MacApp," which draws material from my article of the same name in the May 1989 issue of *Apple Direct.*

The single most important new feature of MacApp 3.0 is its support of several key pieces of System 7 that you should be putting into your programs. In some cases, MacApp 3.0 does virtually everything for you, and in others it simplifies the work you have to do.

Current Events. Here's some good news: A MacApp 3.0 program automatically handles the four Apple events that Apple requires all System 7 programs to implement—Open Application, Open Document, Print Document(s), and Quit. If your program can get by with only these four Apple events, MacApp 3.0 will allow you to strike one important item from your list of things to do.

Of course, the more Apple events your program uses, the more useful it will be in a world of cooperating applications. User scripting is in the future for every Macintosh computer, and each Apple event represents one action that a script (or another program) can ask your program to do. So the more Apple events your program responds to, the more useful it will be.

What does MacApp 3.0 have to do with all this? The MacApp code contains a main event loop and code that intercepts events and sends them to the right object in your program. Much of your work in creating a MacApp program is writing routines—most commonly called *methods* or (in C++) *member functions*—that do the needed work when MacApp hands them an event. (Strictly speaking, MacApp sends an event message to a list of event-handling objects called the *target chain.* An object either accepts the event message or passes it on to the next object in the target chain. When an object accepts a message, it executes the method that is associated with the message.)

MacApp 3.0 treats an Apple event just as it would any other event. The object executes its DoAppleCommand method when it receives an Apple-event message. To implement incoming Apple events, you need only write the code that handles the event and add it to the appropriate object's DoAppleCommand method. (One parameter of a DoAppleCommand message is a command number. The method's code uses it to determine what code to execute.)

MacApp 3.0 also makes it easy for your program to send Apple events. You encapsulate the Apple event you want to send into an object that inherits from a subclass of TClientCommand. To send the Apple event, you simply post the command object to the event queue (using the PostCommand message), as you would for any other event. MacApp 3.0 uses the DoIt message (which you've written) to take care of sending the message and retrieving any reply. (There are two kinds of Apple events that expect replies. With a *synchronous* event, MacApp sends the event and waits for a reply. With an *asynchronous* event, MacApp sends the event and then continues with other work. It keeps track of the pending event and services the reply whenever it returns.)

One limitation: MacApp 3.0 does not support the Apple-events object model**,** because the latter was defined long after MacApp 3.0's design was frozen. (You need this object model only if you are adding sophisticated Apple events to your program. With it you can do things such as ask a document for the third word of its second paragraph.) However, Apple engineers are adding support for the events object model to the next incremental release of MacApp.

Publishing Made Easy. The System 7 Edition Manager gives your documents the ability to *subscribe* to parts of other documents and *publish* parts of their own. (The great thing about publishing and subscribing is that if the original contents—called a *publisher*—are changed, the Edition Manager notifies all the documents that have a copy of that material—each copy is called a *subscriber.* Usually, that notification results in the subscriber updating its contents. The part of your document that contains the actual data of a publisher or a subscriber is called a *section,* and the separate disk file that contains the contents of a section is called an *edition.* For more details on this topic, see my article “Weighing In—Apple Events and the Edition Manager,” on page 8 of the November 1990 issue of *Apple Direct.*)

MacApp 3.0’s support of the Edition Manager is a good example of how you can use MacApp to implement many parts of your program: You create something based on an existing MacApp class and override (write your own code for) selected methods, and MacApp does the rest.

Here are the main steps of adding Edition Manager support to your program. (I’ll leave some of the details to the MacApp 3.0 documentation—in particular, the “Working with the Edition Manager” chapter of *Programmer’s Guide to MacApp.*) First, you make your document a subclass of TEditionDocument and initialize it (this takes a few lines of code). After that, you override methods (or write new ones) to do the following:

- Write a section of your document to an edition file (Do- WriteData).
- Read a section from an edition file into your document (DoReadData).
- Notify the Edition Manager when you change a section (UserSelectionChanged).
- Define designators (used to delineate a selected part of your document).
- Define adorners (discussed below) for your publishers and subscribers. MacApp 3.0 makes this task easy by supplying you with two classes, which are called TPublisherAdorner and TSubscriberAdorner. Both of them are subclasses of the TSectionAdorner class.

Once you’ve done these things, MacApp 3.0 takes care of the rest. For example, it automatically handles the incoming Apple events that relate to sections, and it adds seven new menu items such as Create Publisher... and Subscribe To... to the Edit menu and handles them.

Balloons of Help. Balloons are comic-strip-style text bubbles that help explain some part of the visible display. As with non-MacApp 3.0 programs, the System 7 Help Manager and the BalloonMaker utility make it pretty painless to add help balloons to menus, menu items, windows, and icons.

MacApp 3.0 makes it easier to add Balloon Help to elements *within* a window. (Without it, your program must track cursor movement and explicitly display the appropriate balloon where appropriate.) Since everything in a MacApp window belongs to one of several views, you can easily associate a balloon with a view simply by giving a view the ID number of the desired balloon. You can do this interactively while creating or changing a view with the ViewEdit utility (described below). Compared with a conventional Macintosh program, a MacApp 3.0 program can deliver more-extensive Balloon Help with less effort.

OTHER FEATURES

C++ Worthy. Apple engineers shocked the MacApp community at last year's MADA (MacApp Developers Association) conference by talking about the possibility of converting MacApp itself from Object Pascal (the traditional language for dealing with the Macintosh and everything related to it) to C++. Since language issues are very "religious" (everybody has a strong opinion), the debate about whether or not Apple should convert to C++ was very, shall we say, spirited.

In the end, the Apple engineers decided to translate *all* the MacApp source code to C++. Among the reasons for doing so are that C++ is more powerful than Object Pascal, C++ is a cross-platform standard (Object Pascal isn't), and C++ is where you'll find a *very* large group of skilled object-oriented programmers.

What does this mean to you? If you're using the Macintosh Programmer's Workshop, you can still single-step through both your code (in C++, Object Pascal, or Object Modula-2) and the MacApp 3.0 source code (which is in C++). Object Pascal and Object Modula-2 users, don't panic! You will be able to create MacApp 3.0 programs in these two languages.

Unfortunately, developers who have been using THINK Pascal and MacApp 2.0 are currently out of luck. Because there are quite a few THINK Pascal/MacApp users out there, Apple is looking at a solution that would allow them to use MacApp 3.0 with THINK Pascal—but they won't be able to step through the (already compiled) MacApp 3.0 code. For the moment, though, if you want to use MacApp 3.0, you must do so in the MPW environment.

The C++ syntax, which is more powerful than that of Object Pascal, allowed MacApp's engineers to simplify both the MacApp 3.0 source code and (if you use C++) your MacApp 3.0 programs. MacApp 3.0 redefines some Toolbox data structures to be "intelligent," thus simplifying the code and making it more readable.

For example, MacApp 3.0 defines a new class, CStr255, that *overloads*—redefines—the + operator (and other arithmetic operators) to behave correctly with CStr255 strings. In MacApp 3.0, for example, you can concatenate two strings by saying

```
String1 = String2 + String3;
```

With generic C strings, you had to write something like

```
if((String2[0] + String3[0]> 255))
        String1[0] = 255;
else

     String1[0] = String2[0] + String3[0];
memcpy(&String1[1], &String2[1],String2[0]);
memcpy(&String1[String2[0]+1], &String3[1], String1[0] -           String2[0]);
```

Obviously, MacApp code that uses C++- classes can be easier to understand.

Behave Yourself. In MacApp 2.0, if you wanted to change a class's behavior dynamically, you had to either create subclasses and use each subclass only when the proper conditions occurred *or* you could add some flag variables to the class and have each method execute different code based on the flag's value. Either solution is inelegant and nonportable (that is, it is "hard-wired" into the code of one class).

MacApp 3.0 offers a better solution in the form of *behavior objects,* as implemented by the new class TBehavior. Once defined, a behavior object (or, more simply, a *behavior*) can be attached to an object. If the object becomes part of the target chain (which determines which object handles an incoming event), the behavior can intercept and handle the event *before* the object to which it belongs gets it. An object can have multiple behaviors that are examined in a set sequence before an event gets delivered to the associated object.

Behaviors are both versatile and powerful. For one thing, they are dynamic—they are invoked based on conditions that are known only when the MacApp program is running. In addition, they are modular, meaning that they can be attached to different kinds of objects (they are not hard-wired into the code of one class). They also allow sophisticated modification of object behavior—the AddBehavior and RemoveBehavior methods can attach them to and detach them from an object, and you can use the SetEnable method to activate or deactivate a behavior temporarily.

As one example of behaviors, MacApp 3.0 defines a behavior called the TKeySelectionBehavior, which attaches to a view and allows users to select a text item by typing its beginning letters (the System 7 Finder does this with file and folder names). Because behaviors are modular, you can attach the TKey-SelectionBehavior to any appropriate view. Once you've done this, the view automatically allows file selection by keyboard typing.

Behaviors are very powerful (see the "Working with Behaviors" chapter of *Programmer's Guide to MacApp*), and Apple hopes that MacApp programmers will create lots of useful behaviors and share them with (or sell them to) the MacApp developer community.

Taming the Print Dragon. Printing has always been one of the thorniest aspects of Macintosh development. Although MacApp 3.0 doesn't completely make the dragon go away, it *does* make the beast far more manageable.

Since the view is responsible for displaying a document's contents in a window, it should come as no surprise that printing in MacApp is also tied to views. For simple printing, you need only attach an instance of type TStdPrintHandler to the view you want to print.

As its name implies, TStdPrint- Handler includes enough standard printing features to meet the needs of many programs. When you initialize a TStdPrintHandler object, MacApp 3.0 automatically adds the Print, Print One, and Page Setup menu items to the File menu, *and* it handles their execution.

The default TStdPrintHandler object prints the entire contents of its view by cutting it into page-sized chunks and printing. You can easily customize page margins and the placement of page breaks by using the TStdPrintHandler class. If you require more control, you can make a subclass of the more general TPrintHandler and add the desired extra behavior. (With complex documents, many MacApp programmers create a new view that looks like the desired printed output. They never show this view in a window but use it solely to create the printed document.)

Just so I don't leave them out, I will mention two other MacApp 3.0 printing services. MacApp 3.0 can help you with screen feedback—that is, displaying in the document's window such things as page breaks, headers, and page numbers. MacApp 3.0 can also help you print only the selected part of a document.

Adornable View. As is so often true in object-oriented programming, you may have code that does exactly what you want it to—almost. For example, you may have a view that draws everything you want but you'd like to add a rectangular border around its contents. Sure, you can wade in and change that view's Draw method, but it would be better if you could leave the existing code intact and encapsulate the desired additions in a separate piece of code. That is what *adorners* are all about. They provide a versatile, modular way to change the way a view draws itself. Here are a few examples of how you might use them:

- Customizing the appearance of controls in dialog boxes;
- Drawing the standard borders that the Edition Manager requires for subscriber and publisher sections;
- Drawing the highlighted contents of a view.

Like behaviors, adorners can make your code easier to read, increase the chances that code can be used in more than one situation, and keep your MacApp code modular.

Adorners also have fields that establish their relative priority. By setting these values, you can control whether an adorner will draw before or after the main contents of a view.

You Can Depend on It. Objects are often related to each other, and in the past, you had to write extra code to ensure that a change in one object caused a change in the other. MacApp 3.0 offers a simpler, cleaner mechanism for implementing this behavior. Any object can declare itself a dependent of another. When your code marks the "causing" object (by sending a method named Changed to it), MacApp 3.0 automatically sends each of the dependent objects a DoUpdate message, thus triggering the code that changes the dependent object.

Gently Down the Stream. Most programs need the capacity to save their documents to disk and read them back in. This means writing one routine that breaks your document into a linear sequence of bytes and saves them to a disk file and then writing another routine to read such a file and re-create your document.

MacApp 3.0 can't save you from the work of creating document-read and document-write routines, but it does reduce the amount of coding you need to do. MacApp 3.0 introduces the TStream class, which defines a generalized way to read objects from and write them to an abstract stream of data.

MacApp 3.0 defines three subclasses of TStream: They are called TFileStream, THandleStream, and TCountingStream. MacApp 3.0 also defines primitive methods that know how to read and write bytes, characters, integers, long integers, points, rectangles, and strings to and from a stream. (These methods have such names as ReadPoint and WritePoint.) You subsequently use these primitive methods to define DoRead and DoWrite methods for every class you use.

Ultimately, you read and write documents by sending them a simple DoRead or DoWrite method, using the desired stream as the message's parameter.

Here's the beauty of using streams: Once you've written your DoRead and DoWrite methods to interact with streams, they don't care what kind of stream they get handed. If they're handed a TFileStream object, then your DoRead and DoWrite methods read from and write to a file (automatically handling all the details of dealing with the Macintosh OS File Manager). If they are handed a THandleStream object, they read objects from and write them to handles in memory.

The third predefined stream class works somewhat differently. With no extra programming effort on your part, the TCountingStream class solves an annoying problem—namely, finding out whether there's enough room to write an object out to disk. First, you send the object a DoWrite message, passing it a TCountingStream object. Then, by sending the *stream* a GetSize message, you can find exactly how many bytes it will take to write the object to disk**.** (A TCountingStream object doesn't store incoming bytes; it just counts them.)

TOOLS

Three things make a good software-development platform: the language, the application framework, and the tools/programming environment. The combinations of MacApp and Object Pascal, C++, or Object Modula-2 are good, but they aren't enough. Here are some tools that help you be more productive with MacApp.

Visible Views. Views are at the heart of MacApp—all the views, drawn in the proper order, show a MacApp document as it appears within a Macintosh window. It should come as no surprise that views are usually stored as resources contained within the finished Macintosh program.

Now if your idea of a good time is reading (and, worse, writing) code like this:

```
ViewSignatureAndClassname
  {'View', 510, "", 'ROOT'...
```

You don't need ViewEdit. You can define all your views as Rez source code (like the above, but pages and pages of it) and convert it—with no mistakes, of course—into MacApp 3.0 'View' resources. (Be aware that the resource type of MacApp 3.0 views, 'View', is different from that of the older MacApp 2.0 'view'.)

However, the rest of us are really glad that ViewEdit exists. Think of this program as an object-based drawing program that works on view resources instead of graphics documents. When you open a file, the window that opens makes any 'View', 'STR#' (string list), and 'TxST' (text style) resources visible. (Since views often contain lists and labels, the newest version of ViewEdit has added the ability to edit 'STR#' and 'TxST' resources.)

Double-click the icon of a view resource, and it expands to become a view-editing window (see figure 1, top). Each element of the view is visible, and you can move it, resize it, or shift it to a different layer of the view. (Subview objects appear “on top of” their parent views; this means that changing the depth of an object also changes its position in the view-object hierarchy.)

You can also add new view objects to the window by selecting the object type (TButton or TStaticText, for example) from a pop-up menu and click-dragging the cursor to specify the position and size of the object.

Double-click a view object, and it opens the View Parameters window for that view, allowing you to set the parameters associated with that view (see figure 1). The view parameters are different for each kind of view. If, for example, you open the View Parameters window for a static-text view, you will find that the text in that view may auto-wrap (or not), that it may erase the screen under it before it draws itself (or not), and that it has a specific string and a certain justification mode. Once you have a view looking like you want it to, you can save your work. Later you can call this view resource from your program by using the view’s ID number.

Mouser MacBrowse. New methods demand new tools, and MacApp is no exception. Much of MacApp’s learning curve derives from MacApp’s dependence on object-oriented programming, and the class/subclass/method way of programming is very different from that of the procedural programming most programmers know. Although any sufficiently large program is difficult to keep organized in your head, object-oriented programs seem to be particularly scattered—classes inherit fields from ancestor classes several levels up, methods are overridden by other methods in distant files, and the links among objects are implied rather than visible.

Enter the MacBrowse class/ method/field browser program. (If you wonder why there’s a picture of a cat in the MacBrowse opening dialog box, it’s because previous incarnations of this browser were called Mouser.) MacBrowse lets you browse through all the source code (both yours and MacApp’s) that constitutes your program.

Before you can use MacBrowse, you must first parse all your source-code files. (MacBrowse needs to read in all the source code to learn the interrelationships of all the classes, methods, and fields.) The MacApp 3.0 source code is all in C++ and needs to be parsed only once, but your source code can be in C++, Object Pascal, or Object Modula-2. These need to be parsed once, and individual files must be reparsed whenever their content changes.

Once all the required parsing has been done, the main browser window comes up, as shown in figure 2 . MacBrowse lets you examine your source code through its invisible nonlinear links. (Can you say “hypertext,” boys and girls?)

Click in a class (the upper left pane of the browser window), and you get a list of all its methods (upper center pane), all the class's fields (upper right pane), and its class-inheritance chain (immediately below these three panes). Click in a method, and you get its source code (bottom pane). If the copy of the file containing the source code is not read-only, you can click in the source-code pane and edit it.

The true power of MacBrowse comes from its ability to show hidden connections and provide information that would otherwise be tedious or downright impossible to get. Using various menu selections and their keyboard equivalents, you can do the following things with MacBrowse:

- Go to the definition of a highlighted method.
- Get additional information (where relevant) on the highlighted item, including all the open on-line documentation files for MacApp.
- Find all the references to a highlighted item.
- Display all the overrides of a method.
- Make a second browser window for a specific class (making it easier to compare two areas of code at the same time).
- Show the code that an inherited method inherits.
- Show all the methods and fields that a class inherits from its ancestor classes.
- Access 411 on-line documentation files.
- Show all the methods (in different classes) that have the same name.
- Show a method's documentation (if any).
- Show the name of the source-code file that contains the currently displayed method.
- Show the names of all the source-code files that have been parsed.
- Show all the code segments (if any) and what methods are in each.

MacBrowse also allows you to add new methods and fields to a class and delete existing ones. MacBrowse cannot add new classes to your program, but it's simple to create a new text file with an empty class inside it. If you want, you can then go back into MacBrowse and add methods and fields to the class.

Builds and Bugs. Two other tools that deserve brief mention are MABuild (that's short for "MacApp Build") and SourceBug. MABuild is an MPW tool that helps simplify the building, compiling, and linking of a MacApp program. SourceBug is a high-level debugging tool that works with MPW to give you source-level debugging in a browser environment. (SourceBug is available on the quarterly E.T.O. [Essentials•Tools• Objects] CD-ROM, from Apple, and will later be available from APDA as a separate product.)

When MacApp is present, SourceBug does much of its work in a MacBrowse-like browser window, and it allows you to inspect the current values of variables.

Figure 3 shows several of SourceBug's many features. The upper left window is the browser window, which shows the source code for the DoIt method of TAboutBoxCommand. The black diamond indicates a breakpoint; you can single-step, step "into," or step "out of" the code, or you can have it run until it hits a breakpoint. The white arrow indicates SourceBug’s current location in your code.

The lower left window displays a stack crawl, which shows (in its top pane) all the methods and routines that brought your program to its present location. The active window on the right shows how SourceBug displays all the fields of a given class.

DOCUMENTATION

MacApp 3.0 will come with excellent documentation and source code:

• *Programmer’s Guide to MacApp* is an excellent overview, with separate chapters on objects, applications, events, menus, the mouse, documents, views, special view types, dialog boxes, pop-up menus, the Clipboard, printing, the keyboard, Balloon Help, the Edition Manager, behaviors, and lists. Each chapter shows examples from working sample-code programs and ends with a single-page summary of the chapter’s topic.

• *MacApp 3.0 Tutorial for C++ Programmers* walks you through the compilation of an empty MacApp 3.0 program named IconEdit. Successive chapters explain different aspects of MacApp programming, each adding a feature or two to the original IconEdit to illustrate the material being covered. (The chapters have such names as “Adding Windows to IconEdit,” “Undoing Mouse Drawing,” and “Printing IconEdit Documents.” The last chapter, chapter 18, is entitled “Where to Go from Here.”)

• *Guide to MacApp Tools* covers ViewEdit, MABuild, and MacBrowse, and it also explains how MacApp uses resources. Even though ViewEdit and MacBrowse seem easy to use, they have important features you might miss if you don’t read this manual.

• Definitive class and method reference information is available on line; you can access it from either MacBrowse or MPW 411.

• Nothing takes the place of example source code in helping you learn a complex system, and MacApp 3.0 delivers six working, nontrivial programs in C++ and four in Object Pascal.

The three programs that are available in both languages are of particular interest. IconEdit is the program mentioned in the tutorial book, discussed above. The Nothing program demonstrates the smallest possible valid MacApp program. In the past, programmers have used the Nothing program as the basis for their MacApp programs. Actually, Nothing is too simple, so the MacApp engineers have written a better program, called Skeleton, which they want you to use when you start a new MacApp program. (So keep them happy, OK?)

MACAPP, NOW!

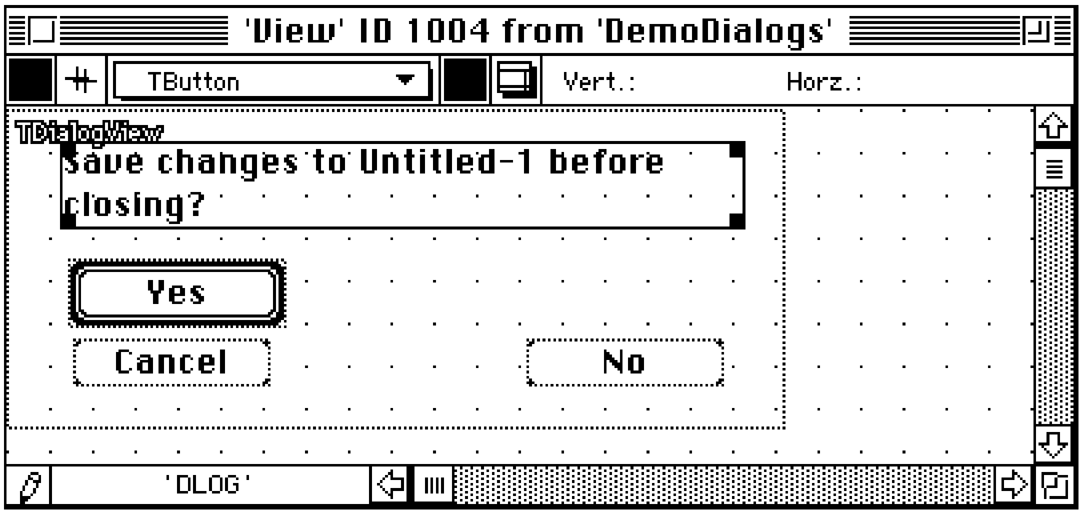
The MacApp platform has never been better. MacApp 3.0 has many powerful new features, including its all-important support for System 7. It completely supports C++, which makes some developers more comfortable with using it. (Previously, some developers didn't want to be tied to Object Pascal, which is not an industrywide standard.) Also, the documentation, tools, and support for MacApp 3.0 make it much easier to learn than MacApp 2.0.

Some of the traditional reasons for using MacApp are even truer today. MacApp takes care of a lot of the code that implements the increasingly complex Macintosh interface. (MacApp also makes it much easier to experiment with and change your program's human interface.) Much of MacApp's code is not just correct but it's also been tested and optimized by years of use, so it's probably better code than you have the time to write yourself. Since MacApp is inherently "blessed" (because it's a product from Apple), its code is the way Apple feels it should be, and it's responsibly written.

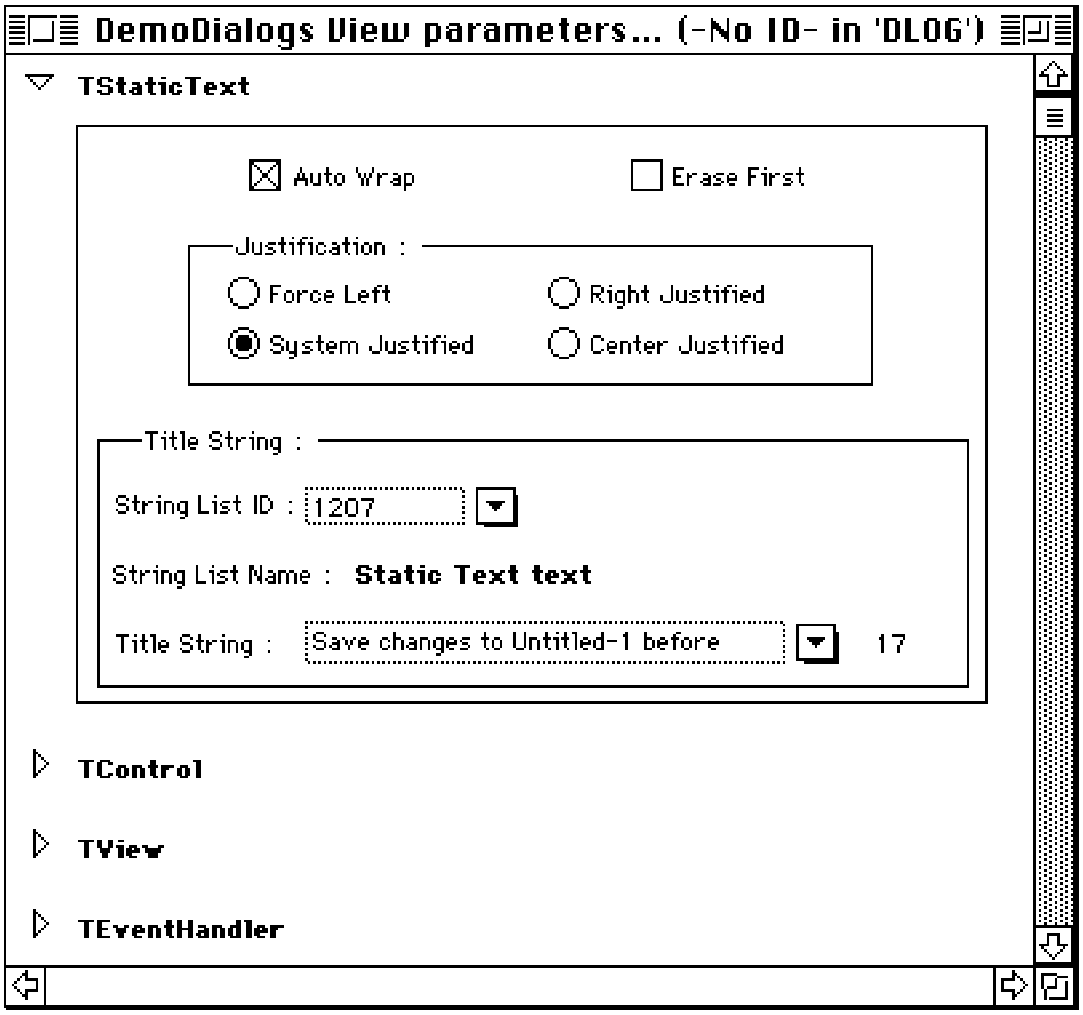
MacApp programs are also less likely to break because of future hardware or system software. Even if they do, recompiling your program with the newest version of MacApp should take care of most problems. And don't forget that by learning to write MacApp programs, you are preparing yourself for programming under Taligent's Pink operating system.

Such major applications as PhotoShop (Adobe), Ray Dream Designer (Ray Dream), Oracle Server (Oracle), ColorSqueeze (Kodak), and more than 40 others were written with earlier versions of MacApp, and more are on the way. MacApp is the state-of-the-art way of creating Macintosh programs, and it deserves your serious attention. ◆

Figure 1: *Editing a view with ViewEdit. The top window shows what a view—here, the view for a dialog box—looks like. The bottom window appears when you double-click on the "Save changes…" static text. This window contains parameters that pertain to the "Save changes…" object.*



'View' ID 1004 from 'DemoDialogs'
TButton
Vert.:
Horz.:
TDialogView
Save changes to Untitled-1 before closing?
Yes
Cancel
No
'DLOG'



DemoDialogs View parameters... (-No ID- in 'DLOG')

TStaticText

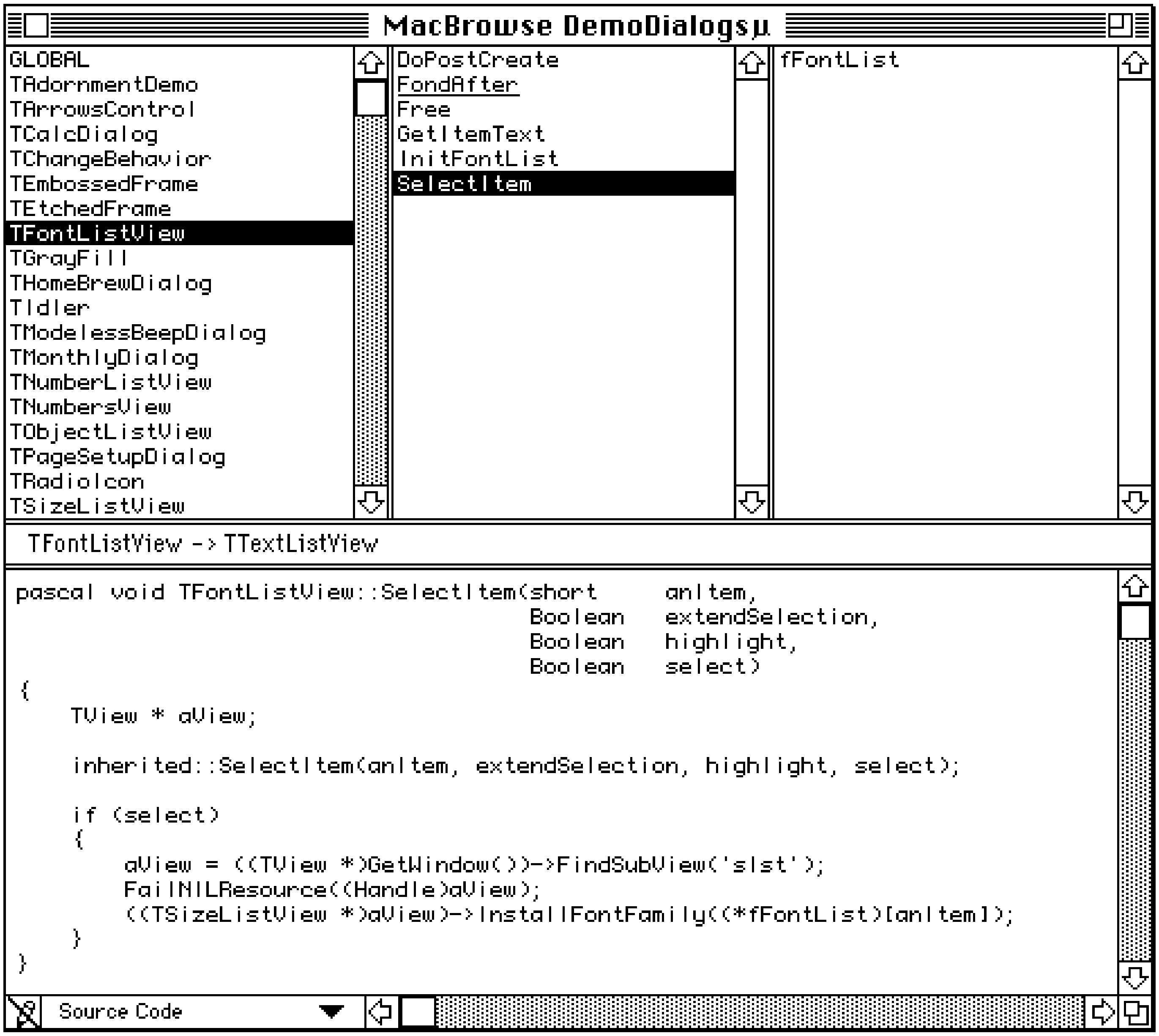
[✓ Auto Wrap] Erase First

Justification :
Force Left
Right Justified
[✓ System Justified]
Center Justified

Title String :
String List ID : 1207
String List Name : **Static Text text**
Title String : Save changes to Untitled-1 before 17

TControl
TView
TEventHandler

Figure 2: *The main MacBrowse browser window. The top three panes show (left to right) classes, methods, and fields, the single line beneath it shows the selected class's inheritance chain, and the bottom field shows the definition of the selected method within the selected class.*



```
GLOBAL
TAdornmentDemo
TArrowsControl
TCalcDialog
TChangeBehavior
TEmbossedFrame
TEtchedFrame
TFontListView
TGrayFill
THomeBrewDialog
TIdler
TModelessBeepDialog
TMonthlyDialog
TNumberListView
TNumbersView
TObjectListView
TPageSetupDialog
TRadioIcon
TSizeListView
```

```
DoPostCreate
FondAfter
Free
GetItemText
InitFontList
SelectItem
```

```
fFontList
```

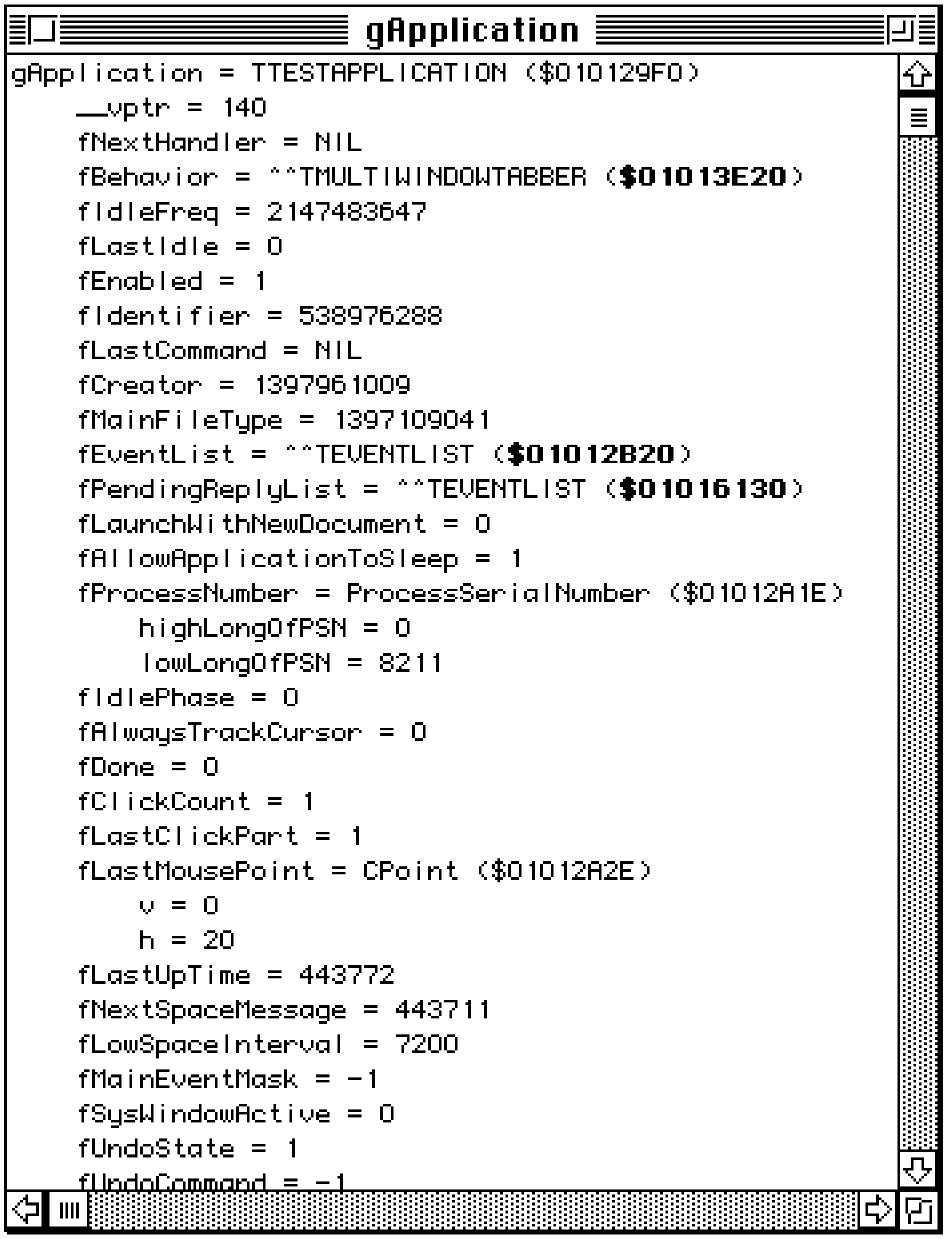
TFontListView -> TTextListView

```
pascal void TFontListView::SelectItem(short     anItem,
                                      Boolean   extendSelection,
                                      Boolean   highlight,
                                      Boolean   select)
{
    TView * aView;

    inherited::SelectItem(anItem, extendSelection, highlight, select);

    if (select)
    {
        aView = ((TView *)GetWindow())->FindSubView('slst');
        FailNILResource((Handle)aView);
        ((TSizeListView *)aView)->InstallFontFamily((*fFontList)[anItem]);
    }
}
```

Figure 3: *Using MacApp with SourceBug. See text for details.*



```
gApplication = TTESTAPPLICATION ($010129F0)
    __vptr = 140
    fNextHandler = NIL
    fBehavior = ^^TMULTIWINDOWTABBER ($01013E20)
    fIdleFreq = 2147483647
    fLastIdle = 0
    fEnabled = 1
    fIdentifier = 538976288
    fLastCommand = NIL
    fCreator = 1397961009
    fMainFileType = 1397109041
    fEventList = ^^TEVENTLIST ($01012B20)
    fPendingReplyList = ^^TEVENTLIST ($01016130)
    fLaunchWithNewDocument = 0
    fAllowApplicationToSleep = 1
    fProcessNumber = ProcessSerialNumber ($01012A1E)
        highLongOfPSN = 0
        lowLongOfPSN = 8211
    fIdlePhase = 0
    fAlwaysTrackCursor = 0
    fDone = 0
    fClickCount = 1
    fLastClickPart = 1
    fLastMousePoint = CPoint ($01012A2E)
        v = 0
        h = 20
    fLastUpTime = 443772
    fNextSpaceMessage = 443711
    fLowSpaceInterval = 7200
    fMainEventMask = -1
    fSysWindowActive = 0
    fUndoState = 1
    fUndoCommand = -1
```

(Slightly) Inside MacApp

MacApp is two things. First, it is a C++ library of object classes, each containing data structures and related procedures, that implement many of the visual and organizational elements of most Macintosh programs. MacApp gives you a set of Macintosh-interface elements that is robust, high-level, and highly optimized. You then write your program in C++, Object Pascal, or Object Modula-2, using the Macintosh Programmer's Workshop (MPW) environment.

But MacApp is not just a library of objects. It is also a framework of source code that provides the foundation for a standard way to write Macintosh programs. Several sample programs supplied with MacApp give you examples to examine. One of them, named Skeleton, is a good starting place for creating your own MacApp programs.

MacApp takes care of all the "usual" details of programs, freeing you to concentrate on the code that's unique to your programs. It doesn't throw out everything you've learned so far about Macintosh programming, but it does replace certain bodies of Macintosh lore.

For example, you still have to know how to draw with QuickDraw, but you don't have to know the details of opening and closing files. When a MacApp program loads or saves a file, it does the opening and closing automatically; your job is to supply the DoRead and DoWrite methods for all the objects in your document. (DoRead and DoWrite are the routines that do the actual reading and writing of your document's contents.)

MacApp's default behavior automatically takes care of any clicks or drags in the scroll bars (also the grow, close, and resize boxes) without your having to write any code. Obviously, though, the contents of your window have to come from somewhere. They come from an object called a *view,* and any view can be used as part of a larger view. A view must know how to draw itself and respond to the events that pertain to it. When MacApp needs to redraw or change a window's contents, it uses that view's Draw method, which calls the Draw methods you have written for each of your views.

A MacApp program interacts with users in three primary ways: through menu events, key events, and mouse events. MacApp handles many of these events automatically (if, for example, you resize a window, select New from the File menu, or type command-Q). But numerous events, such as clicking on an object or selecting a menu item, must be interpreted by the code you write. So how does MacApp know which object should interpret which event?

The short answer to this question is that MacApp gives each event to appropriate event-handling objects until it finds the right one. The long answer involves knowing about the *command chain,* a linked list of event-handling objects that MacApp examines in turn until one accepts and handles the event.

You must define three kinds of command methods to handle events. The command methods are called DoMenuCommand, DoKeyCommand, and DoMouseCommand. (Incoming Apple events, if any, trigger a fourth method, DoAppleCommand.) You implement each of these in the classes that need them. For example, the code for executing a Save a Copy As... menu command is located in the document's DoMenuCommand method, whereas the code for executing a Magnify menu command is located in the DoMenuCommand method of the window's view.

Command classes are subclasses of the MacApp TCommand class; you need one for each kind of undoable event. Your DoWhatever Command method creates and initializes a command object (an instance of the command class) when it responds to an undoable event. You also need three methods—DoIt, UndoIt, and RedoIt—for each command object. DoIt carries out the desired action, UndoIt restores the state of your data to what it was just before the event, and RedoIt redoes an action that has been undone.

MacApp handles an undoable event as follows. When the DoWhateverCommand executes, it creates a command object for the event and places it on the command chain. When MacApp sees the command object, it immediately sends it a DoIt message. Thereafter, MacApp sends the command object UndoIt and RedoIt messages whenever the user executes the Undo and Redo menu items. When the user goes on to other work, MacApp sends the command object a Commit message (to make the most recent Undo or Redo message permanent) and deletes the command object. ◆

MacApp Support

MacApp is more than six years old, so it now has a solid base of technical information and support behind it. Here are some items you should know about:

• MacApp Developers Association is an independent association that supports MacApp with a bimonthly magazine called *Frameworks* and an annual conference (February 24–28, 1992, in Orlando, Florida—if you hurry, you may be able to make it!). Membership is currently $75 per year (U.S.), $95 (Canada), or $110 (all other countries); this figure includes a year’s subscription to *Frameworks.* Contact MADA at 4327 Rucker Ave., P.O. Box 23, Everett, WA 98206, USA; phone (206) 252-6946; the electronic mail address is MADA through AppleLink or America OnLine.

• Apple Developer University teaches a five-day “MacApp and Object-Oriented Programming” class at various sites. (Two other courses it offers are “Language-Independent Object-Oriented Design” [two days] and “C++ for the Macintosh” [three days].) Contact the Apple Developer University Registrar at (408) 974-6215.

Developer University also offers a related CD-ROM-based course called “Introduction to Object-Oriented Programming: Self-Paced Training Course” (it covers OOP in general but not MacApp). This self-paced course covers the same material in both Object Pascal and C++; it costs $295 (U.S.) and is sold through APDA. See the bottom of the “Now Available from Apple” box in this issue of *Apple Direct* (in the "News" folder) for APDA’s phone numbers.

• On AppleLink, there are several group addresses to which you can subscribe. (Everybody in the group gets all the group’s messages, including contributions from third-party developers and Apple employees.) One group address is MACAPP.TECH$; it’s about MacApp in general. The other is currently named MACAPP3BETA$; the name may change, but it will still be concerned with MacApp 3.0 issues. To join either of these, send a request to AppleLink address MACAPP.ADMIN.

A third, related group address, about C++, is named CPLUS.DEV$. To join this group, you have to send your request to CPLUS. ADMIN.◆

MacApp 3.0 Availability

As this issue went to press (late January 1992), all the MacApp 3.0 code is final except ViewEdit. This means that this not-quite-final version of MacApp 3.0 will be distributed on the E.T.O. #7 CD-ROM, to be shipped to subscribers in March. Once ViewEdit is finished (probably second quarter, 1992), APDA will ship MacApp 3.0 as a stand-alone product.

E.T.O. #7 will contain a new version of MPW C++, version 3.2. This version is based on the C++ Language System, Version 2.1, from AT&T Unix Systems Laboratories. The code in this new version of C++ is final, but the documentation is not. This means that, like MacApp 3.0, MPW C++ v.3.2 will be on E.T.O. #7 but will not be available as a separate APDA product for a few months.

In case you haven't noticed, you really should subscribe to Apple's E.T.O., which is a one-stop, one-price source for Macintosh development tools. Once you've bought all the pieces, you can get quarterly E.T.O. CDs for a reasonable yearly fee. (Check with APDA for current E.T.O. pricing.)

If you're going to use MacApp 3.0, the software you need—Macintosh Programmer's Workshop, MPW C, MPW C++, MPW Object Pascal, and MacApp—costs almost as much as E.T.O. does, and if you already own some of the pieces, E.T.O. costs less. Not only does E.T.O. keep you up to date on Macintosh programming tools but it also gives you first access to selected programming tools before they are available separately from APDA. To contact APDA, see the bottom of the "Now Available from Apple" in this issue of *Apple Direct* (in the "News" folder) for APDA's phone numbers. ◆

QUICKTIME GOES CROSS-PLATFORM

Movie Toolkit announced, Windows “player” shown

Developers have been clamoring for Apple to provide more cross-platform tools, and Apple is responding to that demand, as evidenced by the recent announcement that it will bring the benefits of QuickTime to other platforms. The cross-platform announcement followed right on the heels of Apple publicly stating its intentions to begin selling QuickTime-based consumer products by the end of the year.

At the Macworld Expo last month, Apple announced the Movie Exchange Toolkit, which allows developers to convert data from other platforms to QuickTime movies. Apple also demonstrated a prototype of a QuickTime “player” for Windows. Also at Macworld, third-party developers showed more than 100 applications that support QuickTime (see the article in this issue entitled, “QuickTime Supporters at MacWorld”).

By going cross-platform with QuickTime, Apple is working toward a goal of enabling developers and users to convert dynamic data created on other computing platforms to the QuickTime Movie file format for playback on Macintosh computers. Apple is publishing the full specifications for the Movie file format, thus providing dev-elopers of cross-platform applications a standard way of exchan-ging dynamic data among com-puting environments.

The QuickTime Movie Exchange Toolkit will consist of utilities, sample code, and documentation to allow the incorporation of the QuickTime Movie file format into non-Macintosh applications. The utilities support popular models of computers running MS-DOS and Windows, as well as Cray, Silicon Graphics, Sun, DEC, and IBM computers. For example, a user creating an animation in an MS-DOS application that supports QuickTime will have the option of converting that animation to a QuickTime movie. The animation can then be shared with Macintosh users over a network or by floppy disk, and can be played back on a Macintosh.

The QuickTime Movie Exchange Toolkit is available from APDA for $79. Also, you can order the QuickTime Developer’s Kit ($195) from APDA, which contains more than 800 pages of printed documentation and a CD-ROM containing the QuickTime extension, utilities, sample code, sample content, third-party-digitizer components, and HyperCard XCMDs. For ordering information about either product, see “Now Available from Apple" in this issue of *Apple Direct*.

In addition to cross-platform data exchange, developers have expressed strong interest in QuickTime playback functionality for platforms such as Microsoft Windows. It would allow them to offer applications on alternative platforms that could not only create QuickTime data, but could also synchronize, play back, compress, or decompress movies created on Macintosh or other platforms.

Apple has developed an early prototype of a QuickTime player for Windows which it demonstrated at Macworld, although the company has not announced how or when the player will gointo production. ♦

QuickTime Supporters at Macworld

During the January rollout of QuickTime at the San Francisco Macworld, more than 75 developers participated in Apple's various comarketing activities. John Sculley featured some QuickTime third-party demos in his keynote address; Apple also sponsored a QuickTime room, where developers demonstrated their products to customers and the press. The list of developers that participated in those activities is below.

More than 125 new QuickTime products, in virtually every product category, were announced or shown at Macworld. If you have a hot product (QuickTime or otherwise) you'd like Apple to know more about, send an AppleLink message to COOL.APP.

A.D.A.M. Software
ACIUS, Inc.
Adobe Systems, Inc.
Aldus Corporation
Alias Research
Alpha Technologies Group
Aris Entertainment
Articulate Systems
Bliss Interactive
Brøderbund
CD Technology
Compact Designs, Inc.
Company Science & Art (CoSA)
Computer Friends
Compression Labs
Data Translations
Davidson & Associates
Deneba Software
DeltaPoint
Digital F/X
Digital Vision
DiVA
Educorp
E-Machines
Event One
Evergreen Technologies, Inc.
Form and Function
Figment Interactive Design
Gold Disk
Imagine That, Inc.
Intelligent System Design
Interactive Media Design
Interactive Solutions
Inter Optica Publishing Co.
Jostens Learning Corporation
Knowledge Revolution

Knowledge Vision
Learningways, Inc.
Lew & Associates
LightSource
Linker Systems, Inc.
Key Curriculum Press
MacroMind•Paracomp
MacLaboratory
Mass Microsystems
Media Technology
Microsoft
MotionWorks
MultiMedia Corporation
National Instruments
New Video Corporation
Odesta
Olduvai
Ottawa Researchers
Pierce Software
ProficomP GmbH
Radius
RasterOps
Sanctuary Woods
Scholastic Software
Software Toolworks
Specular International
Storm Technologies
Strata
SuperMac Technology
Systat, Inc.
Tom Snyder Productions
Tulip Software
Virtus Corporation
Vividus
Voyager Company
Warner New Media
Wayzata Technology
Wings for Learning
Wolfram Research
WordPerfect Corporation
Workstation Technologies
****

Now available from Apple

The following list shows which APDA products have become available to developers within the last several weeks. To get a full listing of all APDA products, check the current *APDA Tools Catalog.* For new-product announcements and the most up-to-date price lists, check AppleLink (path—Developer Support:Developer Services:Apple Information Resources: APDA—Tools for Developers).

If you're interested in the latest version numbers of all Apple system software products, check "Latest Rev" in the Information Resources folder on the current Developer CD. Latest Rev also tells you where to obtain these system software products. In addition, the "Developer CD Highlights" section on page 3 of this issue tells you which new system software releases appear on the current CD.

Apple Products

AppleShare Server 3.0 API Developer's Kit
R0177LL/A$99.00

Macintosh Development Tools Advisor, 1992 Edition, CD Version
R0171LL/A
$15.00

Macintosh Development Tools Advisor, 1992 Edition, Disk Version
R0124LL/A
$10.00

QuickTime Movie Exchange Toolkit
R0190LL/A
$79.00

Third Party Products

Language Systems FORTRAN v.3.0 with MPW
T0064LL/C
$595.00

Language Systems FORTRAN v.3.0 without MPW
T0065LL/C
$495.00

p1 Modula-2 v.4.3
T0377LL/C
$345.00

Smalltalk/V Mac v.1.2
T0197LL/B
$199.95

APDA TOP TEN SELLERS*

1. DAL v. 1.3 MVS/VTAM Server
2. E.T.O. Starter Kit & Subscription
3. MPW C v. 3.2 bundle
4. MacTCP v. 1.1 License & Developer’s Kit
5. DAL v. 1.3 MVS/TSO Server
6. APDA Technical Information Mailing
7. Macintosh Common Lisp v. 2.0b1
8. Inside Macintosh, Vol. I- VI + X-Ref
9. MPW C++ v. 3.1
10. MPW Development Environment v. 3.2

*As of 1/16/92

To place an APDA order from within the U. S., contact APDA at (800) 282-2732; in Canada, call (800) 637-0029. If you need to call the U. S. APDA office from abroad, the number is (408) 562-3910. You may also send an AppleLink message to: APDA. If you’re outside the U. S., you may prefer to work with your local APDA contact. For a list of non-U. S. APDA contacts, see the “International APDA Programs” page in the *APDA Tools Catalog.*

SOFTWARE PROTECTION IN CHINA

The Chinese government has signed a Memorandum of Understanding agreeing to comply with international standards on foreign interventions.

According to the report in the *New York Times*, the agreement may help reduce the estimated $419 million in royalties lost annually in China because of illegal copying of books, music, and computer software.

In short, it appears that this Memorandum of Understanding will improve protection of developers' software in China.

CD Highlights, Feb. ’92

This column is your guide to the latest Developer Series CD, entitled “20,000 Leagues Under the CD.” It tells you what’s new and notable.

To quickly access everything listed below, see the “What’s New on This CD” folder (located at Dev.CD Feb 92: Start Here/Read CD License lst:What’s New on This CD?). Here you’ll find aliases to every new and updated package on the disc, including those highlighted below. The February CD includes the following highlights:

System 7 Tune-Up: This month’s disc includes the new release of System 7 Tune-Up, a set of system modifications designed to improve the performance of System 7.0 and 7.0.1.

It improves performance in low-memory situations, printing, file sharing, and networking.

AE Registry/Winter ’92: The *Apple Event Registry: Standard Suites,* Winter 1992 Edition has been completely redesigned to be easy to read and use. (For more information, see the “Apple Event Registry” news item on page 1 of this issue.)

Tools Advisor 1992: The Tools Advisor is an electronic guide to Macintosh development tools. This HyperCard stack includes tool data sheets, stories about how the tools have been used in particular projects, and definitions of various technical terms.

Developer University Demos: This month we feature demos of three self-paced courses offered by Developer University: Macintosh Programming Fundamentals, Introduction to Object-Oriented Programming, and AppleTalk for Programmers.

Snippets: To speed your searches, we have reorganized the Snippets folder by subject. Take a look, and tell us how you like it! To see the latest Snippets (short samples of code), go to the What’s new on this CD? folder in Start Here. Aliases to this month’s new additions are located there.

***develop* magazine**. The February issue is in the Periodicals folder. It contains the following articles:

• *Making the Most of Color on 1-Bit Devices,* a two-part article that describes how to create color PICTs on black-and-white machines and explains the theory and practice of dithering.

• *The TextBox You’ve Always Wanted* describes a replacement for TextBox that provides better performance and more flexibility, as well as international compatibility.

•*Making Your Macintosh Sound Like An Echo Box* explains how to use double-buffering techniques to simultaneously record and play sounds.

•*Simple Text Windows Via the Terminal Manager* explains how the Terminal Manager (in the Communications Toolbox) provides handy text-output capabilities in your application—with virtually no effort.

• *Tracks: A New Tool for Debugging Drivers* is about a new debugging tool that provides an easy way to log information from your driver, greatly easing your debugging woes.

Tools Demos: This hot-off-the-press package includes demonstrations of Macintosh Common Lisp, Virtual User, MPW, and MacApp.◆

PACIFIC MARKET TAPES AVAILABLE

If you weren’t able to attend the Apple Pacific Market Forum, held January 16–17 in San Francisco, you can catch up via audiotape. Most sessions were recorded and are available for purchase.

The forum, which featured Apple executives and industry ex-perts, focused on market-entry and expansion opportunities in Japan, Australia, Canada, Latin America, and the Far East. It provided a venue in which developers could discuss distribution, financial, and partnership issues.

The conference included three tracks that allowed attendees to tailor participation to their specific needs: country/regional market specifics, market entry, and market expansion.

Sessions dealt with such topics as the challenges of cross-cultural communication, how to establish overseas operations, enhancement of products with media integration, and Macintosh system software advantages for overseas markets. They also included developer and distributor success stories.

A highlight was the multimedia information kiosk, designed to help participants meet potential business partners.

Apple Pacific, which sponsored the event, also inaugurated the Step Ahead Awards (known as the Steppies). Seven awards were presented to developers whose products have helped to create new markets in the Apple Pacific region. Overall, more than 500 people representing over 20 countries attended, including developers, distributors, republishers, consultants, and Apple employees.

Audiotapes of the keynotes, sessions, and forums are available. To request a list or to place an order, write to Mobiltape Company, Inc., 25061 W. Ave. Stanford, Suite 70, Valencia, CA 91355; phone (805) 295-0504; fax (805) 295-8474. Individual cassettes cost $9 each, and the entire set of 19 tapes costs $189.95 (plus tax and shipping/handling). Discounts are offered for some larger quantities. ◆

A View(er) on the Paperless Office

Dear Tog,
As we slog through the endless snows of winter, a question arises, the answer to which seems to lead to pseudoinfinite human-interface recursion (ouch!). We're very interested in your thoughts on this one. Here's our dilemma:

In designing a tool that allows viewing of any file (for now, let's suppose we limit it to text files, as opposed to graphics files), we come to an interesting decision point. Since we do not want to write a word processor, we stipulate that the window we display will be read-only—that is, its contents cannot be edited—and that the user can drag, select, and copy any text from the window.

Furthermore, as we display files from various word processors, we will preserve the format (font, size, style, indents, and the like) for users to see. Hence, users are able to view documents created by a word processor they don't have, the operative word here being *view,* not *edit.*

Here's the rub: Suppose a user opens a file only to find that it was created in 2-point Courier and cannot be read without an ultrahigh-resolution monitor and a scanning electron microscope? We'd like to be able to offer a choice of fonts and sizes, in case these tools don't happen to be handy. But then we'd be editing, since changing the font and/or the size really is editing the document—but we said that we didn't allow editing.

Further, suppose that this document with the 2-point Courier also has some 12-point Times in it and the user selects 10-point Helvetica from our font and size menus. Do we change all the text? Do we change just the selected text? Uh-oh, now we're editing again. We don't support editing, just viewing. What if the user has selected nothing?

In summary, we want to provide users with the capability of viewing any document. We'd like to let them change font and size, but we don't want to support editing. Suppose for a second that we ignore the fact that changing these is editing. How much of the text do we change?

We'd appreciate some of your human-interface wisdom on this one. Short of writing a universal word processor, it just doesn't seem as if there's a way out that leaves users with intuitive options. Got any ideas?
—Ford Rackemann,
Digital Equipment Corp.

Tog responds: Historically, viewers have been handed a finished product. The reader of a modern paperback novel has no more control over the font size and style than the peruser of an ancient Egyptian papyrus did. This has led us into a belief, based purely on a lack of a counterexample, that readers not only cannot but also should not be able to affect the document they are reading.

Of course, we do have a few specialized counterexamples, in the form of interactive products that enable the viewer to follow various pathways through a body of material, but for the mainstream of electronic documents, the rule that the writer "edits" and the viewer "reads" remains unchanged.

The "paperless office" was heralded with great fanfare more than ten years ago. Since then, nothing has happened.

Our offices are still knee-deep in paper, and they will remain so until the advantages of an electronic document outweigh the disadvantages of having to insert such a document into a computer in order to be able to view it.

Figure 1 shows what will happen with the paperless office concept if we do nothing to help it along. The gray “mountain” represents the slowly decreasing difficulty-of-use experienced by users of electronic documents today. The black bars represent the current small advantage of electronic documents over paper ones. Since this scenario assumes we do not increase the advantage, the black bars remain the same height. The ultimate acceptance of electronic documents is represented by the emergence of the black bars into the light at the end of the graph.

Some of the mountain in the way of acceptance of electronic documents includes lack of standardized file formats, lack of portability of computer equipment, and the difficulty of reading on today’s electronic displays.

Already, people such as you, Ford, are attacking the file format problem, and in the future lie smaller, lighter, less-tethered, and more easily-viewed computers. But we do need to wait for computers as handy as a paperback book? No. We can accelerate the acceptance of paperless documents by offering people solid advantage.

Figure 2 demonstrates how quickly and thoroughly electronic documents could be accepted and even embraced if we took certain steps to increase their value over their paper counterparts. And the first step we can take is to give users more control over the viewing process.

ACTIVE VIEWING

Today, we have special classes of documents—applications, really—that enable viewers to alter both the appearance and presentation order of the original document. Generally, these systems—labeled interactive—are designed for the selective retrieval of information from large bodies of data, for educational purposes, or for entertainment.

Full-scale interactivity is the extreme of viewer control. It comes at a high price for the creator of the original material: It requires oodles more time to create interactive systems than to view them. But there can be a lot of intermediate stopping-off points between zero user control and the massive user control of interactive applications.

Readers’ Rights and Font Control. It is appropriate and reasonable to enable a viewer to alter the size and style of the original fonts, at least for the purpose of easing the reading process. Indeed, it is unreasonable to do otherwise, not only because of the example Ford mentioned of the 2-point type, but also because many people with limited vision cannot read 12-point type!

One could argue that the writer intended the readers to have to squint to view the 2-point type and that it is thwarting the writer’s artistic intention to enable resizing. I would argue that such a writer is a foo-foo head, unworthy of rights.

We are not talking colorization here, where third parties intercede between author and reader to impose their own style on the original. Indeed, the only question of ethics applies to what happens when the reader tries to save the font changes. The changes should be stored in a separate resource so that the writer's original document remains pristine, there for instant reversion, should the reader so wish.

In answer to Ford's question, what happens to the 12-point Times when the 2-point Courier grows to five times its original size in the conversion to 10-point Helvetica? At least four things could happen:

- The change could have no effect on the 12-point Times.
- The 12-point Times could change to 12-point Helvetica.
- The 12-point Times could become 60-point Times (12 x 5 = 60).
- The 12-point Times could become 60-point Helvetica.

I would say that of all these, the only one that makes sense is the first. If users want everything in 12-point Helvetica, let them do a Select All from the Edit menu and then choose the new font. If they want everything to change in scale, let them use the Command/Shift/< and Command/ Shift/> key combinations after selecting all. (Some programs already use these combinations to change font sizes proportionally: 2 becomes 3, and 12 becomes 13. Actually, 12 can jump directly to 14 or some other larger size, depending on the program design.) Better yet, give them some content-zoom buttons in the status area at the lower left corner of the window...

Relative Fonts. The other option I would explore is based on my relative-color idea, covered in chapter 37 of that fine book *Tog on Interface,* available from purveyors of quality books and at select computer boutiques ($26.95—cheap). Relative fonts would enable users to change the specific fonts being read without changing the author's overall aesthetic design. A user could change the main typeface used in the document, for example, and the viewer application would select accompanying typefaces that would maintain the author's original intention. If the user chose a sans-serif font instead of a serif font, the viewer application, upon discovering that all the headings were in a sans-serif font, might switch them to a serif font, thereby maintaining the author's assumed intention to have headings be of an opposite style from that of the main body of the text.

Relative fonts would additionally prevent the ridiculous: Changing the 2-point type to 10-point type would not change the 24-point headings to 120-point type (24 x 5). Instead, it might leave the 24-point type alone. As for the 12-point type in this circumstance, it might move up to only around 16. In effect, relative fonts would cause the range of type sizes to compress rather than letting fonts grow to an unusable degree. (Of course, it would have to be defeatable, in case someone really wanted 120-point type.)

Midlevel Interactivity. Why stop with font control? With the advent of universal readers, writers will want to take advantage of the electronic format by including interactive elements within their documents. That does not mean that they will want to create the kind of elaborate interactive systems we have today. Rather, they will want to enable users to get at extra information that costs the writer little to provide, such as the underlying data that supports a given chart or perhaps the original documentation from which the writer's report was drawn. For an in-depth view of how this midlevel interactivity might work, I recommend the lengthy discussion in chapter 37 of *Tog on Interface* (on sale at finer cosmetic counters everywhere).

ULTIMATE CONTROL: THE AUTOMATIC PLAGIARIZER

When I went to school, we did our term papers the old-fashioned way: We strung them together out of passages copied out of various books and encyclopedias, with vocabulary suitably modified. Sometimes it worked, sometimes it didn't. I remember one student who was exposed as having copied an article on California history in its entirety out of a journal that only two people in the world ever read: the student and his teacher. The poor fool of a student had left the vocabulary virtually untouched, and the teacher, recognizing a put-up job when he saw one, spent three days in the library until he found the original.

I predict, in the not-too-distant future, the arrival on our college campuses of automatic plagiarizers, able to analyze our more honest scribblings and then applying our personal grammar, style, and vocabulary levels to information that has been gleaned by agents from electronic searches of CD-ROM discs, network databases, and the Library of Congress. Student users will claim the automatic plagiarizer to be the calculator of the next century; their professors will take a darker view. This is ultimate reader control, and like the majority of ultimate extensions, it will force us to extend our ethics as well.

Until next month, my AppleLink address remains TOG. ◆

GET NEXT EVENT

Here are the trade shows/events at which Apple Computer, Inc. is scheduled to exhibit as of press time. This list may be incomplete.If you have information about a show you want listed here, write to Apple Direct Event Calendar, 20525 Mariani Avenue, Mail Stop 75-2B, Cupertino, CA 95014. For further information, check the Events folder on AppleLink (path—Developer Support:Developer Services:AppleInformation Resources: Developer Events).

March 9 through 12
NCGA - Nat'l. Computer Graphics Assn.
Anaheim, CA
Contact: Martha Filson
(703) 698-9600

March 10 through 12
CD-ROM Conference and Expo, San Francisco, CA
Contact: Cahners Exposition
(203) 964-0000

March 18 through 21
SPA, Seattle, WA
Contact: The Conference Dept. of SPA
(202) 452-1600

March 26 through 29
NSTA - Nat'l. Science Teachers Assn.
Boston, MA
Contact: NSTA
(202) 328-5800

March 29 through April 1
NAESP - Nat'l. Assn. of Elem. Sch. Principals
New Orleans, LA
Contact: Kim Hardage
(703) 684-3345

March 30 through April 2
FOSE - Federal Office Systems Expo
Washington, DC
Contact: National Trade Productions
(703) 683-8500

April 4 throug 7
ASCD - Assn. for Super.
& Curriculum Dev.
New Orleans, LA
Contact: ASCD
(703) 549-9110

April 8 through 10
Macworld Expo
New York, NY

Contact: Mitch Hall & Assoc.
(617) 361-8000

April 8 through 10
Macworld Expo
Stockholm, Sweden
Contact: Torgny Husén
(46) 8-6679180

April 13 through 16
SCOOP West
Contact: Suzanne Dinnerstein
(212) 274-0640

April 15 through 18
NBEA - Nat'l. Business Education Assn.
Boston, MA
Contact: NBEA
(703) 860-8300

April 21 through 23
EUC Conference (European University Consortium)
Bruges, Belgium
Contact: Veerle Vandereecken
32-2-741 22 11
Fax: 32-2-741 22 42

April 23 through 25
The Mac Show, Tampa, FL
Contact: Libby Barland
(215) 540-9111
Fax: (215) 628-0882

April 23 through 26
Macintosh Consultants Conference
Boston, MA
Contact: Mac Consultants Network
(209) 545-0569

April 25 through 28
NSBA - Nat'l. School Boards Assn.
Orlando, FL
Contact: Teresa Dumouchelle
(703) 838-6722

April 28 through May 1
Computer 92, Lausanne
Contact: Kathrin Mäder
1/832 81 11 (Switzerland)
Fax: 1/830 63 06

May 3 through 8
IRA - Int'l. Reading Assoc.
Orlando, FL

Contact: IRA
(302) 731-1600

May 5 through 8
Worlddidac, Basel
Contact: Kathrin Mäder
1/832 81 11 (Switzerland)
Fax: 1/830 63 06

May 12 through 16
IFABO, Vienna A-1020
AppleLink: A.MARCOM
or BONENGL.L

May 13 through 15
GTC West, Sacramento, CA
Contact: Dave Draxler
(916) 452-4902

*****

So What’s Your New Product All About?

Position It With the Elevator Test

by Geoffrey Moore,
Regis McKenna, Inc.

I got the call at 3:30 P.M. on a Friday. It was a slow day at Regis McKenna, Inc. My feet were up on my desk. I was thinking (we think a lot at RMI). Then the phone rang.

“Mr. Moore, you don’t know me, but I attended the Software Publishers Association conference at Orlando where you spoke on positioning, and I’d like to ask you some follow-up questions, if you don’t mind.”

After exchanging a few pleasantries, I learned that my caller was a Cambridge-based software developer of PC as well as Macintosh products, with an “information product”—his words—that he was ready to release. Now that the product was complete, he was interested in looking at marketing issues.

“No time like the present,” I quipped (we quip a lot at RMI). “What can I do for you?”

“Well, first of all, everyone tells me that the key to success with this product is to position it correctly, but I’m not exactly sure what all that entails.” I told him to put his feet up and spent the next few minutes explaining the importance of positioning.

The Importance of Positioning

1. Positioning is both a noun (as in the position/space something occupies in peoples’ heads) and a verb (something you do to influence the position you get in peoples’ heads).

2. The most important thing to know about positioning as a noun is that the space you get in other peoples’ heads is very, very small—typically no more than a phrase or two.

3. The most important thing to know about positioning as a verb is that if you attempt to pack the positioning space in people’s heads with more information than it can hold, the space overflows, your “entry permit” is canceled, and your entire message is rejected as noise.

“But that’s impossible!” spluttered my caller. “With a new type of product like ours, you can’t possibly explain everything in a phrase or two.”

“True, but since your customers don’t want to know everything, so what?”

“So how do I know what to tell them?”

I explained that this was what consultants charged money for, and he explained that he was a bit short of funds at the moment, having just been through a painful, expensive, and unrequited acquisition attempt —too painful to discuss. I could relate to that; I had just priced a new car myself. Anyway, he asked if I could just elaborate a little on how one went about solving this type of problem.

The Elevator Test. “Have you ever heard of the elevator test?” I asked.

He had not. I explained that the elevator test worked as follows: You and another person get on an elevator. As you press the button for, say, the 14th floor, the other person asks, “So, what exactly is this new product of yours all about?” If you have answered this question to the other person’s satisfaction before the doors open, you have passed the elevator test.

This test is one that venture capitalists like to apply to new companies as well as new products—before they fund them. They know that the success of any new venture depends in part on a successful word-of-mouth communication campaign. They figure that if the developer cannot explain things in a few words, then no one else is likely to be able to either and any kind of consistent word-of-mouth communication will therefore be doomed to failure. So it is *crucial* to pass the elevator test.

“Wait a minute!” interrupted my caller. “In the first place, my building doesn’t have 14 floors—so I can’t even practice this test. Besides, you haven’t helped me at all. You’ve just restated my problem.”

Drat. This guy was too savvy for me. I was beginning to suspect that he had been a consultant himself in some prior life.

“OK, OK, I’ll give you some of the basics. But this is my lifeblood I’m giving away” (we do not normally give away our lifeblood at RMI).

How to Pass the Elevator Test

Basically, I asked him to draw a simple *x,y* graph, and then label the *x*-axis Key Benefit and the *y*-axis Key Differentiation. I drew the graph (see Figure 1) while we talked, hoping that he was doing something similar at the other end of the phone.

The simplest way to think about the positioning space you get in peoples’ heads, I explained, is to break it up into two components: the Why buy? component (the *x*-axis, Key Benefit) and the Why me? component (the *y*-axis, Key Differentiation). You get only one “sound byte” for each axis.

“But there are seven or eight great reasons to buy this product!” exploded my caller.

“Great,” I replied, “but for any one type of audience, pick one. Throw the other seven away. Otherwise all you will generate is noise.”

“But for any one benefit, there’s all kinds of competition,” he moaned.

“That’s what the *y*-axis is for. You get one shot at differentiating yourself from all that competition. Again, why one?”

“Because people won’t listen to more than one,” he intoned, his voice pitched halfway between evident pleasure in anticipating my point and the ensuing depression of realizing its implications. “I get it. I get it. But give me an example.”

I thought for a minute back to a recent project we’d done to help a developer introduce a new monitor. The product, I explained, is a video-display monitor that can pivot between landscape orientation
(11" x 8 1/2") and portrait orientation (8 1/2" x 11"). The idea is that for spreadsheets and presentations, landscape is better, whereas for letters and documents, portrait is preferable. So why not just switch whenever you want?

“How does it work?’ he wanted to know.

“Never mind how it works,” I replied. “The question is, how do you position it?”

“Oh, yeah. OK. How did they position it?” I talked him through the diagram in Figure 2.

There are actually quite a few reasons a person might want a such a monitor: It looks cool, it saves desk space, it lets you switch back and forth in real time; but the developer decided that for its prime target customer—an administrative support person—the main attraction was getting a page-at-a-glance image for documents (portrait) as well as presentations (landscape). They called this benefit Full-Page Display.

Now, it turns out that with displays that do not pivot—landscape-only displays—you must have at least a 19-inch monitor before you can see a full 8 1/2 x 11-inch vertical page. With a pivoting monitor, you can get the effect with a 15-inch display. That gives this new monitor a major cost advantage. Hence, low cost was chosen as the key differentiation.

“But what about the pivot itself?” he asked. “Surely that is the key differentiation.”

Well, yes and no, I answered. Yes, pivoting is a unique and memorable differentiating feature. But no, it is not the key differentiator, if what we mean by that term is the basis on which customers will prefer your product over its competition.

Customers are rarely comfortable allowing a “unique” feature to be the basis for determining a competitive choice. Hence Tandem’s early problems in positioning fault-tolerant computing, and the problems positioning and finding a niche for the current pen-based computers.

In this context, the pivot—alone, only as a pivot—is no more than a gimmick. But the pivot as the key to a low-cost implementation of full-page display is the basis for a competitive preference.

“Well, that’s all well and good. That company has something tangible to sell. But we’ve got a software product that is part data management, part data selection, part data display—plus we supply the data to go with it, along with our commitment to update it on an ongoing basis. We’ll never get it down to anything this simple.”

“Look, this is all in my book, chapter 6, pages 155 and following—so if I tell you all this, will you promise to go out and buy it?”

He said he would, and I made him memorize the ISBN number, which my mom has had stenciled on the inside of all my underwear.

____

Creating a Positioning Statement

OK, here is a method for capturing your positioning strategy in a simple, two-sentence formula. The formula is as follows:

For	<target customer>
Who	<compelling reason to buy>
Our product is a	<product category>
That	<key benefit>.

Unlike <main competitor>

Our product <key differentiation>.

"Whoa, slow down. What's this all about?" he asked.
Positioning, I explained, is based on winning the battle for mind- share in any given market segment. A market segment is in part defined by specification of a target customer and a compelling reason to buy. Change the target customer or the reason to buy, and you change the segment you are attacking.
"Why compelling reason to buy?"
I explained that in a high-tech market, if the reason to buy is not compelling, then it is all too likely that potential customers will not buy at all.
"So in the case of the pivoting monitor," he said, "the target customer was an administrative assistant. But what was the compelling reason to buy?"
I replied that the target customer was actually the administrative assistant's boss, the one who would have to approve the product's purchase. And the compelling reason to buy was to improve productivity on a task that made up the bulk of their relationship with each other.
The statement, in other words, should begin like this:

For executives with administrative assistants **who** need to generate documents and presentations frequently and with rapid turnaround, **our product is** a video-display monitor **that** provides a full-page display regardless of whether the page orientation is landscape or portrait. **Unlike** any other VDT that provides this capability, **our product** costs thousands of dollars less.

"So what is this —the advertising copy?" my caller asked.
Not at all, I replied. This is the positioning statement. As such, it governs all positioning activities related to the product. It has an affect on not only advertising but also public relations, sales presentations, brochures, demos, and any other form of marketing communication. That is, the statement sets the criteria for accepting or rejecting ad copy, press releases, and the like. If the copy is not "on strategy," as defined by this two-sentence statement, then no matter how catchy it is, it is not acceptable and will be less likely to sell the product.
"But most importantly," my caller interjected in a voice bright with enthusiasm, "this will let me pass my elevator test."

"Right," I said.
"Now, if I can just keep an executive staff long enough to come up with a good statement..."
And with that reply, my caller hung up.◆

Geoffrey Moore is a partner at Regis McKenna, Inc., a Palo Alto, California firm that specializes in marketing strategy development and market relations services for technology-based clients. He is the author of the book Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers*.*

Figure 1:

Positioning Space Defined

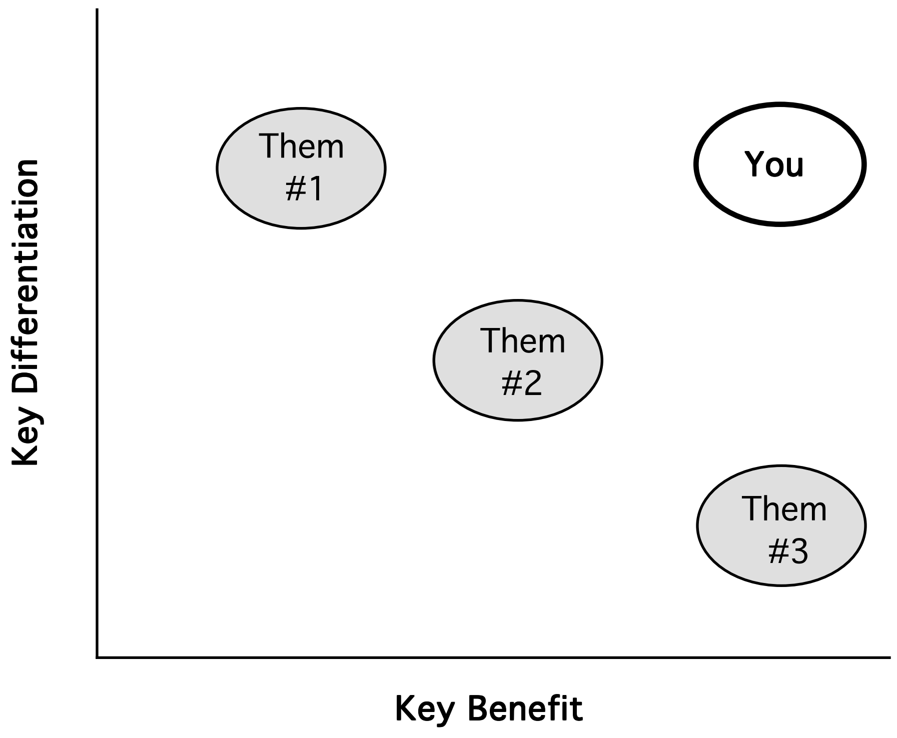
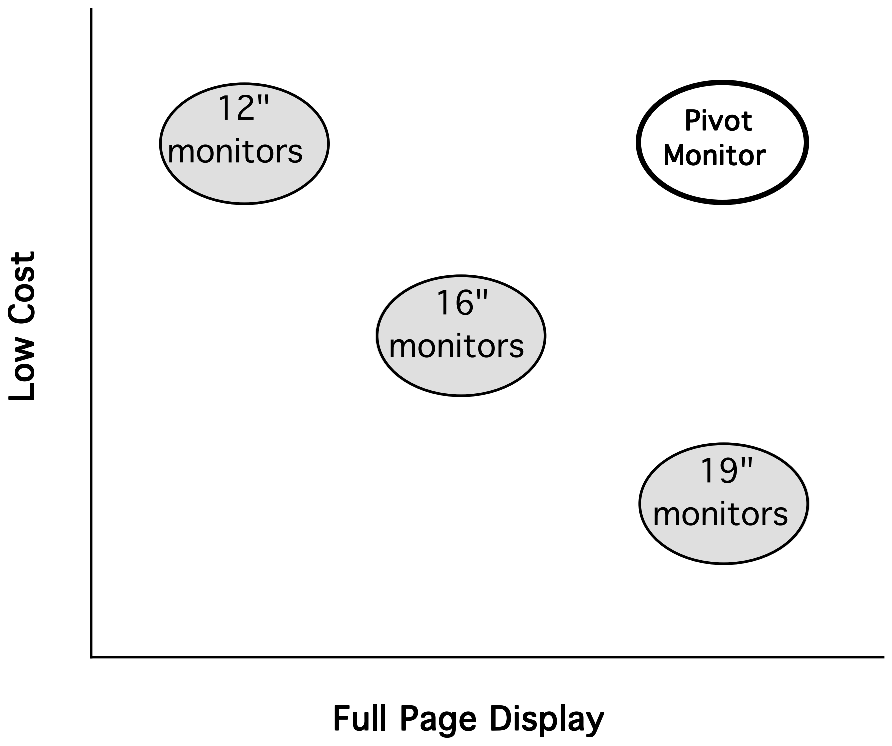


Figure 2:

Example: The Pivoting Monitor



How We Practice Customer Focus

Doing Right By The Customer

by Linda Itskovitz, Intuit

“If you think your customers are bothersome, just try doing without them for 30 days.” I’m not sure who said it or if those were the exact words, but that’s what stuck with me. Without customers, we’re nothing. That’s easy to say, but the difficult part is to put your money where your mouth is.

To be truly customer-focused, you must believe that the most important element in being successful is doing right by the customer. You can never be satisfied with doing 99 percent of the job; instead, you have to give the customer 110 percent. That means spending the time, money, and other resources to find out what “doing right by the customer” means to *your* customers and then putting it into practice.

Generally speaking, it means (a) learning who your customers are, (b) learning what they need and like and why they buy, (c) understanding what satisfies them, (d) testing your premises, and (e) applying the results to your products and services.

Although these steps may seem obvious, it does take a certain mind-set, focus, and infrastructure to put those principles into practice. In addition to having a product customers need and like, there are four other things that help keep us customer-focused:

Being market-driven. This means that the development of products is driven primarily by what the market demands, not just by the desire to implement the newest, hottest technology. Development- and/or technology-driven companies certainly can be successful and can have a strong customer focus; but being market-driven gives you an edge, because you don’t have to shoot in the dark—that is, your products and marketing programs are driven by market research. Our development labs create products based directly on the customer/market research we do.

In this respect, high-tech companies can learn from some of the more traditional consumer-products companies. I feel that Intuit is unique in the industry, because it has a heavy consumer-products orientation and approach.

Several Intuit executives and marketing people have backgrounds in consumer products, marketing everything from hair spray to kitty litter. They have a lot of experience in market research, expertise in tracking the results of their programs, and a knowledge of sophisticated merchandising techniques. They also tend to have a strong dedication to their customers and are experienced at communicating a product’s benefits to them.

Basing decisions on data. An important factor in developing a customer focus is having a philosophy about how to make product and marketing decisions. One way is to base them solely on subjective judgments, which often isn’t the wisest choice. On the other side of the coin, decisions are based purely on data.

Basing decisions on data is the cornerstone of Intuit’s corporate decision-making process. Although we recognize that not all decisions can be made this way, our goal is to reduce the gap between those based purely on judgment and those made with the available data.

We're trying to assure that the controlling factors guiding us are customer and market information. Therefore, it is important to gather as much data as possible so that we have a sound, complete foundation on which to base decisions.

We not only use the more tradtional sources of market intelligence, but we also go to great lengths to get direct input from
our customers.

Putting decision making into the right hands. We think that it's necessary to empower those who are closest to the marketing and customer data to make major product and business decisions. In our organization, the product managers are closest to the data and the customers, so they make many of these decisions.

Intuit views product managers as business managers. (This concept is based on a model successfully employed by some Japanese car manufacturers and other consumer-products companies.) Our product managers are charged with defin-ing product needs, shepherding the research and development team, and generally galvanizing the company.

Their goals: to elevate products to a dominant market-share position, to increase the size of the market and market penetration, and to do all this at a reasonable cost. They have the leverage and the resources—and the motivation—to do what needs to be done to meet these goals.

Although everyone in the company is charged with being focused on the customer (and everyone makes a significant contribution to customer satisfaction), the people who are closest to the big picture are the product managers—and they make the decisions that have the most influence on the product and our customers.

Making sure that the customer focus permeates the entire company. Our philosophy is that customer satisfaction is the charge of every employee, no matter what they do in the company. That means that everyone must learn what our definition of customer satisfaction is and what it means specifically to their respective jobs. One good way to do that is to put every employee in direct contact with customers. To do so, we've instituted a telephone contact program (see "Reach out and touch them," below), in which every employee has ample opportunity to speak directly to users.

PUTTING THE PHILOSOPHY INTO PRACTICE

Intuit has implemented several programs to get input from customers and, at the same time, communicate to them how important their input is. Here are just two of our programs.

Reach out and touch them. Our telephone contact program is a critical element in our customer-focus efforts. Everyone in the company, including the CEO, is required to spend time each month talking with customers on the phone. (Imagine the surprise of the customers who find out that they're having their concerns addressed directly by the firm's CEO!)

This program is so important that we've instituted a formal process to make sure that everyone participates regularly. We take this process very seriously, to the point of making those who miss a month do double-time the next.

The telephone contact program has been in practice since the company was founded. It started when the company was very small; because the organization had only a few employees, each person had to get involved in many activities just to get the product out the door, from product box packing, to answering phones, to doing telephone product support.

Now this practice serves as a constant "real life" reminder of why the company exists, and it keeps us focused on customers. Because it is a key priority in everyone's job, all employees come to understand just how important our customers are.

Follow Me Home. Our Follow Me Home program is particularly useful. As the name implies, we visit new customers' homes (with their permission, of course) and watch their first experience with our products.

We ask stores that sell our products to give us the names of customers who are willing to participate. Then everyone on the product team visits customers' homes and literally watches over the customers' shoulders as they install and use our product.

We ask the customers to talk us through what they are doing and what they are thinking as they work with the application. We take notes and assess the situation: What do customers have problems with? What features seem to be easy or difficult to understand? Our main goal is to study the product in users' environments. (This process takes time; we have spent as many as five hours in a customer's home.)

One thing is sure: We learn a lot each time we make a visit. Some of the valuable lessons we've learned include the following:

•*Users don't always read what's on the screen.* For example, we thought that it would be helpful to put a message at the bottom of the screen that prompts the user about what to do next; but by watching and listening as customers used the product, we saw that they often didn't notice the prompt. The lesson learned: You can't just throw something onto the screen and expect users to see it.

•*Consistent grammatical structure is a must.* For instance, the main menu of the DOS version of Quicken used to read:

Register
Write/Print Checks
Create Reports
Select Accounts.

The items on the menu were not grammatically consistent, so many customers thought that "Register" meant "register the software" instead of "use Register," which was what we had in mind. Therefore, we changed the menu to read "Use Register."

•*Users don't look at all the menu options before they begin using a program.* One assumption we had made was that before using an application, users would look through the various menu options to see what the program could do.

None of the customers we observed did this. They just started using the product right after they installed it.

Customers have been very receptive to participating in the program. From their point of view, our desire to get their input demonstrates that we truly care, that we are interested in their problems and needs, and that we value their comments.

THE FUTURE CHALLENGE

Intuit continues to grow, which we feel is a reflection of our concern for customer satisfaction. However, our growth also creates a challenge: how to keep a very strong customer focus as we broaden our product line, and as our organization becomes larger and larger (we now have 300 employees).

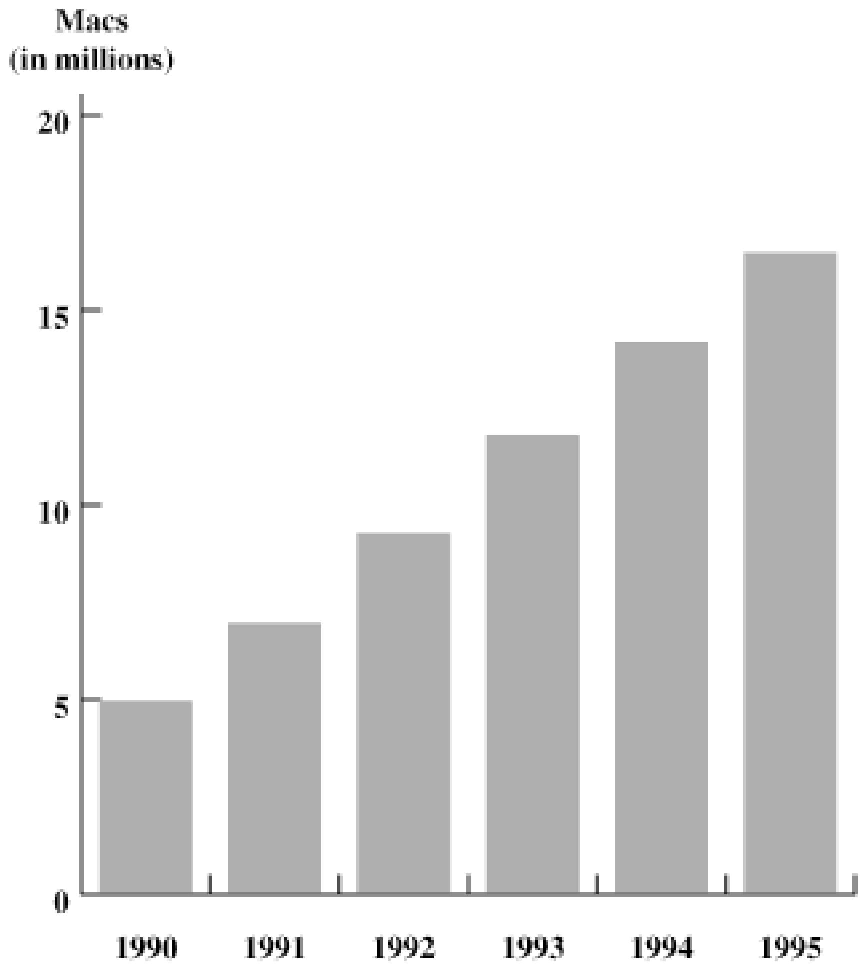
When the company was smaller, it was easier for everyone to stay abreast of what was happening in the entire company and of all of its products. Now, the sheer size of the company and the complexity of managing multiple products on multiple platforms makes it harder for individuals to know much beyond the scope of their own jobs. We'll need to continue putting the internal programs and resources in place to make sure all employees stay close to the customers, and to ensure that our customer philosophy continues to permeate the entire company.

Also, while we have been changing, our customers are also doing so. The ongoing challenge will be to keep our finger on the pulse of our customer group and make sure that the pieces are in place to give them our all—and keep them 110 percent satisfied.◆

Linda Itskovitz is a Quicken product manager for Intuit, a personal finance-management software company based in Menlo Park, California.

Market Insight

Mac Installed Base Exceeds 6.9 Million



International Data Corp. (IDC), a third-party market research firm, estimates that the worldwide installed base of Macintosh CPUs exceeded 6.9 million units in 1991, and forecasts growth to more than 16 million units by 1995. IDC also estimates that in 1991, non-U.S. markets accounted for 44 percent of the installed base of Macintosh CPUs, and that by 1995 that number will grow to 52 percent of the total installed base.—**Source: IDC**

Apple Activities at SPA Conference

“Strength in numbers. There’s never been a better time to join the revolution.” This will be Apple’s theme at the upcoming SPA (Software Publishers Association) Spring Symposium, to be held March 18–21 in Seattle, Washington. This is an opportunity for developers to learn more about how they can take advantage of Apple’s market-share growth.

Apple will host a suite where you can meet evangelists and Apple solutions-marketing managers to learn about development tools, the Apple developer support program, important technologies, and developing on the Macintosh.

We will also raffle three PowerBooks during the course of the conference. If you’d like to try one, you can participate in our “PowerBook for a Day” loan program. Apple will host the Business Center (your office away from the office) at the conference, and will participate in several conference sessions. The company will also offer an update about its antipiracy efforts.

The registration fee for association members is $650 prior to February 7, and $775 afterthat date. For nonmembers, the fee is $950 before February 7 and $1,075 after. For more information, call the SPA at (202) 452-1600. ◆

It Shipped!

Through the "It Shipped!" program, you can announce new and revised third-party products in *Apple Direct.* It Shipped! listings are also made available on the 3rd Party Connection AppleLink bulletin board. You can obtain an It Shipped! application by downloading it from the AppleLink network (AppleLink path—Developer Support:Developer Services:Apple Information Resources: Developer Program Information:It Shipped! Program). Or contact Todd Luchette at (408) 974-1241 (voice) or (408) 974-3770 (fax). Once you've completed the application, send it to Engineering Support, Apple Computer, Inc., 20525 Mariani Ave., M/S 42-ES, Cupertino, CA 95014; Attn: It Shipped! Program. Or send it via AppleLink to IT.SHIPPED.

Optionally, you may wish to send us a copy of your product to be placed in the Engineering Support Library, where it may be checked out by Apple's testing groups for compatibility testing, or by research and development employees for evaluation. If you would like your product(s) to be included in the Engineering Support Library, send them to the address above. The following products shipped in December 1991:

Publisher	**Product (Version)**
U.S.A.	
The AG Group, Inc.	LocalPeek (1.1)
	Net Watchman, Jr. (1.0)
Attain Corporation	In Control (1.0)
Brio Technology, Inc.	DataPivot (1.0.1)
	DataPrism (1.7)
Calico Publishing	Desktop Pattern (1.1)
Coral Research	TimeLog (1.0)
Coyne Company	MacHam Radio General (1.0)
ISIS International	System 7 Pack (1.0)
Neon Software	NetMinder Ethernet (2.0)
OCCAM Research Corp.	MUSE (1.0)
Paradigm Infosystems, Inc.	ProRata Calculator (1.0)
	RiskManager (1.2)
PeCox (Software) Ltd.	Call Me! (1.0)
Sheridan Software	Stitch-It (1.0)
Strategic Mapping, Inc.	Atlas Pro (1.0)
Texas Caviar, Inc.	Vital Signs: The Good Health Resource (1.0)
Vertical Solutions	FastComm (1.0)
	FastLabel (3.1)
	MacLabelPak (1.0)
Wyoming Software	Business Sense (1.5)

CANADA

Advantage Software, Inc.	PatchMaker (2.0)
Folkstone Design, Inc.	Anatomist: A Human Anatomy CD-ROM (2.0)

Correction
Wallpaper 1.0, which was mentioned in It Shipped! in the January 1992 issue of Apple Direct, *is a product of Thought I Could. It appeared incorrectly as being a product of T/Maker Company. Our apologies!*

Developer Resources

Get answers to many of your development questions, expand your knowledge, or find help for your programming projects by tapping into these resources.

Developer Associations and User Group Programming Special Interest Groups (SIGs)

These groups can provide programming resources such as informational meetings, bulletin board services, technical tips and techniques, and contact with other developers to help with your development projects.

DEVELOPER ASSOCIATIONS

•**MacApp Developers Association.** Dedicated to supporting users of MacApp. For information, call (206) 252-6946.

•**SPLAsh Developers Association.** Dedicated to supporting users of Symantec programming languages. Call (415) 527-0122.

• **Macintosh Scientific and Technical Users Association.** Provides members with a forum for the exchange of information on Macintosh computer use in scientific and engineering applications. Call (508) 755-5242; AppleLink: CONS.LAB.MFG.

• **United Kingdom Developer Council.** Dedicated to helping support any Apple UK developer. Contact Paul Smith, telephone: Henley (0491)574295; AppleLink: PGSMITH.

•**Austrian Macintosh and Developer Association.** Contact Klaus Matzka, Ungargasse 59 A-1030, Vienna, Austria. AppleLink: AMDA.

•**The Netherlands - VAMP (Vereniging Actieve Macintosh Programmeurs - Association for Active Macintosh Programmers).** Contact VAMP, P. O. Box 11029, 2301 EA Leiden, The Netherlands.

•**Argentina - Get Info, Association for Argentinian Macintosh Developers.** Contact Jorge Sagasti, Zapiola 1625-1249, Buenos Aires, Argentina. Telephone: (541) 551-4990.

USER GROUPS

General. To locate an Apple User Group Programming SIG in the U.S., call (800) 538-9696, x500. For information on user groups outside of the U.S., contact your local Apple office.

• **Berkeley Mac User Group Programming SIG.** For membership and benefits information, call (415) 849-9114.

• **Boston Computer Society Mac Tech Group Programming.** For membership and benefits information, call (617) 625-7080.

• **Germany User Group Programming SIG.** Contact Mac e.V., SIG Programmieren, Herr Thomas Kramp, Krimmstr.25, 6750 Kaiserslautern.

• **Japan User Group Programming SIGs**

MacFarm. Contact: Nobuhiko Inage
4-107 Kamimachi, Yokosuka-shi, Kanagawa-ken 238
Tel: 81-468-22-4352

SMAC (Shikoku Maniac & Apple Club)
Contact: Masaaki Oyama
736-3 Shinmyou Takase Mitoyo Kagawa 767
Tel: 81-875-72-1188 (only Japanese); Fax: 81-875-72-5140

Tama Macintosh Users Group. Contact: Atsuo Mutoh
Nitta Labs Tokyo Univ. of A&T,
Naka-cho Koganei, Tokyo 184
Tel: 81-423-81-4221 (EXT. 433)

JUG-CPM.SIG-Mac. Contact: Satoru Tsuzuki
3-8-3-504, Hikarigaoka, Nerima-ku, Tokyo 179
Tel: 81-3-3976-8179; Fax: 81-3-3976-8179

*****

APPLE DIRECT

Apple Direct is Apple's monthly developer newspaper, covering business and technical issues for decision makers at development companies. It is published by Apple Computer, Inc.'s Developer Support Systems and Communications group.

Publication team:

EDITOR-IN-CHIEF:
Dee Kiamy (AppleLink:KIAMY)

TECHNICAL WRITER/EDITOR:
Gregg Williams (GREGGW)

PUBLICATIONS AREA MANAGER:
Hartley G. Lesser (H.LESSER)

FORMATTER:
Lisa Ferdinandsen

CONTRIBUTORS:
Jessa Vartanian, Suzanne Dills, Sharon Flowers, Monica Meffert, Suzanne Forlenza, Kris Newby, Cindy Cain, Tammy Gniffke, Katherine Parsons

MANAGER, DSSC:
David A. Krathwohl

CONTENT GROUP MANAGER:
Greg Joswiak

FILM:
Aptos Post, Aptos, CA

PREPRESS:
Prepress Assembly,
San Francisco, CA

PRINTER:
Wolfer Printing Co., Inc.
Los Angeles, CA