The Electronic Magazine for Macintosh™ Developers

Issue #3

Part I

(Second Release)

7/5/85

Contents

This issue uses the following fonts: New York 9, 10, 12, and 18 point; and Geneva 9 and 10 point. If you are going to print the magazine, these fonts and the sizes twice as large should be present.

MacDeveloper is in no way sponsored by or associated with Apple Computer, Inc.

MacDeveloper is published the first Friday of each month. Distribution is via electronic bulletin board systems and national information services. If these avenues of distribution are not accessible to you, send a self-addressed, stamped envelope with a Macintosh diskette to the following address; unless you request a different issue, the latest issue of MacDeveloper will be returned.

Harry Chesley
1850 Union Street, #360
San Francisco, CA 94123

Special notice to BBS operators: You are actively encouraged to post this magazine on your BBS, so long as all of it is posted. The larger the distribution of MacDeveloper, the more articles and advertising we will get to support the magazine, to everyone's benefit.

Read This!!!

This is the third issue of MacDeveloper. From here on, it either becomes a viable commercial enterprise, generating enough revenue through paid-for advertisements to finance the authors' and editors' time, distribution costs (long distance phone calls), etc., or it fades into the sunset.

The reason this article is titled **Read This!!!** is that MacDeveloper's future is, to a very large degree in your hands, and for a very simple reason: advertisers only pay if they're convinced someone will actually see the ad they're paying for. So we need to convince them that MacDeveloper is read by more than ten or twelve people crazy enough to spend the time to download it.

On the next page is a questionnaire. Please take two minutes to fill it out and mail it to me. This is the only way to check the circulation of MacDeveloper — unlike most magazines, which keep track of the number of copies printed (what does that mean with electronic distribution?) and the number of subscriptions.

You can fill in the questionnaire on-line, using MacWrite, and then print it by using the print command and specifying that it print only page 4. Or, you can print out the page — or photocopy it if you have a printed copy — and fill it in by pen or pencil. Once you have the questionnaire filled in and printed, put it in an envelope and address it to:

Harry Chesley
1850 Union St., #360
San Francisco, CA 94123

Then put a 22 cent stamp on it and mail it. Consider it a 22 cent subscription to MacDeveloper.

Even if you get this issue some time after the publication date, please send in the questionnaire. The fact that the magazine is still being read a month or more after publication is also an argument that it's a good place to advertise.

For BBS operators, there's also a second questionnaire on page 5; if you're a BBS Sysop, please fill in both. The point to this second questionnaire is two fold: (1) I'm compiling a list of BBSs which have Macintosh messages or files; and (2) I want to get an idea of how MacDeveloper gets around, so that I can optimize the distribution.

If you have any additional comments, please send them too.

Remember, **only** if enough of you send in this questionnaire will you see another issue of MacDeveloper!

Harry Chesley

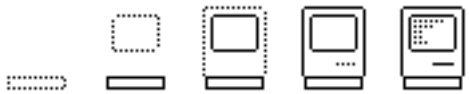
All readers, please fill in the following (name and address are optional):

Name:
Company:
Street:
City, State:
Zip code:

- Where did you get MacDeveloper from?
- How did you get it (modem, disk, photocopy...)?
- How many people get a copy from you?
- How do they get it (modem, disk, photocopy...)?
- What telecommunications services do you use?
- How many hours per month do you use them?
- What telecommunications software do you use?
- What is your occupation?
- What do you use the Macintosh for?
- What does your company do/produce?
- How many people are there in the company?
- If you or your company develop software:
  - How many people are involved in Mac development?
  - What development environment/compiler do you use?
  - Do you decide on the environment/compiler for your company?
- What do you like best about MacDeveloper?
- What would you like to see changed?
- What was your favorite article?
- What was your least favorite article?
- Additional comments:

BBS Sysops, please fill in the following:

- What is your BBS's phone number?
- What is the BBS's name?
- What hours is it in operation?
- What speed(s)?
- What computer is it running on?
- What BBS software do you run?
- What is the orientation of the BBS?
- Is the BBS access restricted (such as a corporate BBS)?
  - If so, what is the restriction?
  - If so, do you want me not to list your BBS?
- Is there a membership requirement/fee?
  - If so, what is it?
- When did the BBS start operation?
- How many callers do you get/month (roughly)?
- How many regular callers do you have (roughly)?
- What percentage of the files on your BBS
  come from which of the following sources (roughly)?
  User uploads:
  CompuServe:
  Delphi:
  Other BBSs (which, if you know):
- Would you be interested in a BBS oriented magazine?
- Ditto a BBS oriented user group?
- Additional comments:



Mac C Me!
A Review of Consulair's Mac C Compiler

Léo Laporte
KLOK-FM Radio
77 Maiden Lane
San Francisco, CA 94108
CIS 75106,3135

Introduction

Let's take it for granted you want to write software for the Mac - for fun or profit. Let's further assume you aren't interested in making an investment in the now defunct Lisa. You need a programming language that will work on a single 128K Mac with two drives.

Given these minimum requirements, there are at least a dozen products that'll do the trick, from Microsoft BASIC to Apple's Macintosh Development System. But, I submit, to really do the job right, any programming language for the Macintosh must satisfy five tougher criteria:

- The language should provide means to access **all** the Macintosh ROM routines.
- It must be capable of producing applications that adhere in every way to the Macintosh user interface standards.
- Finished applications should not require a run-time module or interpreter (that is, they should be able to be run stand-alone from the finder).
- The language should have some sort of interface to assembly language.
- The syntax and structure of the language should be close enough to Pascal to allow a reasonably knowledgeable programmer to use **Inside Macintosh** without having to make extensive translation.

Consulair's Mac C Compiler fills the bill in all these respects. I've been using it

full-time since November, '84, and although it's not perfect, Mac C is an excellent programming tool.

Mac C and MDS

The Mac C compiler translates C source text into 68000 assembly language. To create an application you must assemble the code, and link the resulting REL file with one of several i/o libraries provided with Mac C. Assembly and linking require Apple's Macintosh Development System (MDS) - an additional $150 expense. Both the MDS and Mac C were written by Bill Duvall of Consulair Corp.

Mac C Specs

The Mac C disk comes with Edit, a multi-window text editor (the same editor that comes with MDS), Exec, a limited batch processing program that automates the compilation process, and, of course, the Mac C Compiler. A second disk contains the run-time libraries.

Mac C is a full C (as described in **The C Programming Language** by Kernighan and Ritchie [K&R]). Floats aren't supported by the basic version (currently 1.56), but for $60 Consulair sells a floating point upgrade that follows Apple's SANE standard.

Mac C differs from other microcomputer C's in several other regards. A double slash ('//') may be used to comment out the rest of a line, structures are always global to the entire source file, and local variables may be declared anywhere in the text of a function before the variable's first use. These and other minor differences are completely documented in an appendix to the Mac C Manual.

If you'll pardon a bit of computerese here - Mac C supports eight data types: char, short int, int, and long int, all signed and unsigned. Short defaults to 16-bits, int and long to 32-bits, but a compile time switch can force int to 16-bits. True register variables are implemented using four data and three address registers. The latest release also supports enumerated data types, structure assignment, functions returning structures, and passing structures by value. All aggregate data types are supported. All data except char's are word aligned.

The Libraries

Two function libraries come with Mac C. The first supports most of the Unix style functions described in K&R, including string operators, printf(), scanf(), putchar(), getchar(), fopen(), and calloc(). As part of its initialization, this library opens a "TTY" window which simulates a C-style stdin and stdout console

device.

This TTY window will prove useful to programmers porting C source code from another computer, and to novices who want to use one of the existing C textbooks. While it's open, you can think of your Mac as a little IBM-PC: no menus, no graphics, just terminal style i/o on a windowless screen. Of course, if you're writing a Macintosh style program you'll find the window of little use, but you can close it to get it out of the way.

There's also a very small (566 bytes) library that provides neither the TTY window nor any of the usual stdio routines. If you're willing to abandon any thought of porting your code to another machine, you can use this tiny library and rely on the Macintosh toolbox for i/o support.

Catch My Signal?

Both Mac C libraries implement three functions that can make error-handling in your program worlds easier: CatchSignal(), Signal(), and LocalSignal(). These functions behave like the ultimate GOTO - Wirth may not like them much, but they solve a real problem in practical programming: handling errors within deeply nested functions.

The first time CatchSignal() is called it saves the current program context and returns zero. Your program can continue normally. If, however, in any subsequent function, Signal() is called with a non-zero argument, control is transferred to the location of the last CatchSignal() call. The context at the time of the CatchSignal is restored and CatchSignal() returns the argument passed to it by Signal(). Your program can then respond to CatchSignal() as if the original call had returned a non-zero argument. LocalSignal() behaves similarly but is for returning to a CatchSignal() call within a single function.

CatchSignal() allows you to get out of a deeply nested state without multiple returns. Since it can lead to non-structured programming, its use may irk purists, but when used solely in the context of error-handling, CatchSignal() really save your hide.

Using the Macintosh ROM

Mac C comes with 20 header files that provide symbolic names for most of the data types and constants used with the Macintosh toolbox. The compiler scores very highly on its support for the Mac ROM. All but twenty or so ROM calls are directly supported, the rest can easily be called by using the #asm/#endasm directive which allows you to include assembly language in-line in your C source.

Calling a Macintosh ROM routine from Mac C is (to quote my Arkansas uncle) as

easy as spittin'. The Pascal examples from **Inside Mac** can be used almost verbatim. The compiler automatically makes the distinction between stack-based and register-based routines and passes parameters appropriately. You'll only need to translate the Pascal data types into C. There is one sticking point, though, and that's the matter of string representation.

The String's the Thing

C represents a string as an array of characters terminated by a null byte. The Macintosh toolbox routines expect strings in Pascal form: as an array preceded by a count byte, and therein lies the difficulty. Some compilers use "glue" routines to convert the C strings automatically before passing them to the toolbox. Consulair lets

the programmer handle her own string translation. I think this is a wise choice, since there are several ways to resolve the string issue, each with its own advantages and disadvantages.

The first, most obvious, way is to convert your strings by invoking a translation function. All the Mac C libraries provide two such functions: CtoPstr() and PtoCstr(). Each performs an in-place conversion of the string constant. For example:

```
CtoPstr("This is a string");          /* convert into a Pascal string */
DrawString("This is a string");       /* call the toolbox routine */
PtoCstr("This is a string");          /* restore to a C string */
```

An alternate method is to include the assembler directive

```
#asm
STRING_FORMAT = 1
#endasm
```

at the beginning of your C source code. This instructs the assembler to store all strings in Pascal format. In effect you've changed how Mac C represents strings.

A third, and, I think, preferred, method of handling the string problem, however, is to follow the Macintosh user guidelines to the letter and keep **all** string constants in a resource file. This neatly sidesteps the issue. On the Macintosh, at least, it's bad form to hardwire strings into your source code.

Unfortunately the latter two solutions make any library functions that expect C-style strings useless. You'll have to write new ones. That's too bad because Consulair has a library full of useful string operations on its Mac C Toolkit Disk.

The Mac C Toolkit

Consulair offers the Toolkit Disk separately, but most Mac C owners choose to purchase the Toolkit with the Compiler. The Toolkit routines come in a huge (8k) library that replaces the standard library and supports things like serial i/o and extended string and file handling. This library is too big for anyone who wants to write lean and mean code, but can save a lot of time and effort if you're trying to create quick programs for personal use.

Some of the Toolkit routines are improvements on the stdio library. Others provide a simple interface to complex ROM-based functions. Still others are just handy utility routines. The functions are loosely grouped into six categories: string manipulation, system utilities, memory management, TTY i/o, serial i/o, and disk i/o. The Toolkit library will be of most use to someone who doesn't want

to spend hours studying **Inside Macintosh**.

What makes the Toolkit most valuable for the rest of us, though, is the source code that comes with it. Studying it is the best way to learn how to do things on the Mac. Source code for **all** the libraries is included, as is the startup code that must be linked in with your application. If you want to customize your system by changing the initialization code, you need the Toolkit. Likewise if you want to modify the library routines, or lift parts of them for your own programs. By the way, Consulair grants its users unlimited use of the libraries, even for commercial use. You need only include Consulair's copyright notice in your documentation. Consulair is to be congratulated on its enlightened attitude toward both licensing of its libraries and the distribution of their source code.

Learning by Example

Consulair also makes an Examples disk available to registered users for a nominal $20. A variety of simple programs are included on the disk with their source code. You'll find a terminal program, a printing program, examples of how to use the Mac's finder information and application parameters, assembly language fragments that show how to create an ICON resource and call the Standard File Package, and much more. This disk is welcome source of example programs to complement **Inside Macintosh.**

The most valuable program on the Examples disk is called DeskMaker. It turns a program written in Mac C into a desk accessory using a script file that you create. Sketchy but adequate documentation is included on the disk. DeskMaker is a quick and dirty application. Its error reporting is virtually worthless. But, despite its warts, it's the most useful programming tool I have.

I'm happy to report Consulair plans to release another Examples Disk soon.

Error Reporting

When an error is encountered in your source code, Mac C opens a file with the extension .CER and records three things: the line containing the error (in context), the error message, and the line number of the error. It will continue recording errors to this file until it reaches the end (or gets thoroughly fed up with the mess you've made of it) at which time it throws you back into the editor (if you're using Consulair's Edit), considerately opening the source and error files in separate windows for your corrections.

An error during compilation often causes a cascade of spurious error messages. Sometimes even the real error messages are less than informative. For example, Edit will insert an invisible character into the text if the Enter key is inadvertently pressed. The compiler chokes on this character, but produces the totally confusing error "Missing ;", followed by a long stream of spurious errors.

Imagine staring at the line

```
int i;
```

and trying to figure out how it could be missing a semi-colon. Eventually I discovered that re-typing the line cured the problem, but after an evening listening to my curses and epithets, I don't think my wife will ever believe me again when I tell her I program for relaxation.

Despite these flaws, the Mac C error messages are no worse than those of most microcomputer compilers, and a darn sight better than many. In the majority of cases they are to the point and helpful. The manual lists all the possible error messages followed by a brief description of the conditions that can cause them and even suggests solutions.

Documentation

The Mac C manual is a spiral bound 7.5" by 9" paperback volume of about 150 pages. It devotes most of its space to describing features unique to Mac C. This is not a tutorial. It is clearly intended as a reference work for an experienced C programmer.

The manual is at its weakest describing the stdio library. The chapter is little more than a list of functions. The reader is referred to K&R for amplification. On the other hand, the Toolkit library is described thoroughly. The manual is a little fuzzy about how to use the different libraries - it took me a month before I discovered the miniature MacCLib - but an experienced user will be able to figure it out eventually. If you're a novice, you may need to enlist the help of a local expert.

The manual shines, though, when it describes the code generation techniques used by the compiler. I've never seen such a complete description of any compiler's behavior in a manual before. If you want to use another language with Mac C you will find this chapter very informative and useful.

There are appendices containing a list of possible error messages, all the Macintosh traps supported by the compiler (and their argument types), and two sample programs. There is no index.

Along with the manual, Mac C comes with a user support phone number. Bill Duvall actually **encourages** his users to call if they are having a problem. Of course, novices would be better off consulting local resources, but if you're one of those people who doesn't like to use a compiler unless you have direct access to its author, Mac C is for you.

A Day in the Life

Writing and compiling an application in Mac C requires some familiarity with the Macintosh user interface, and a working knowledge of how language compilers work.

You can create your source code using any editor that can produce plain text files. I use the Edit program that comes with the compiler and highly recommend it.

If you have the MDS you can compile your program in one step directly from Edit by using the Mac C Exec. The Exec executes a JOB script that automates the compile process from beginning to end. You can also use the Transfer menu to call the Mac C compiler, or exit to the finder and run it from there.

Mac C has a variety of compile options, some of which may be selected from the Mac C menu, others of which are set with flags in your C source code. The menu options allow you to choose between treating warnings (pointers pointing to different objects, etc.) as errors or not, creating more or less verbose error files, and creating a token file or not. The token file is a version of your C source with comments removed and macros expanded.

Another menu switch allows you to include your C source as comments in the ASM file that Mac C is generating. This will prove useful for programmers who want to optimize at the assembly language level, or for anyone who wants to see what Mac C is doing with his source code. This is one capability I always look for in a C compiler (and, unfortunately, don't always find).

Besides these options, there are several others you can control from your source code using the #options directive.

You can tell Mac C not to convert char and short function arguments to int, to change the default integer size from 32-bits to 16, and whether to force word alignment in structures. There is also an option that will suppress the conversion of trap names during compilation. Only function names beginning with '#' will be treated as Mac trap calls. I'd say this option is included for historic reasons, since it produces some incompatibilities with the header files when it is set.

You can also set the various symbol table sizes with the #options directive. Mac C uses six symbol tables: Global, Typedef, Field, Local, Struct, and Type. Each is allocated space depending on the total memory available. On a 512K Mac almost 170K is dedicated to the symbol tables, more than enough for the most overblown program. It's easy to exceed the limits on a 128K Mac, though - even a modestly sized program can be too big to compile if a large number of header files are #included - so you can use #options to shoehorn your program into the Mac's limited memory.

Once Mac C has converted your C source into assembly language it automatically calls ASM from the MDS to complete the compilation. The linker converts the assembled files into an application. Since it integrates with the MDS, Mac C

supports MDS tools like RMaker. In fact, creating an application with Mac C, is very much like creating an application with the MDS: if you like the latter, you'll like the former.

The Case of the Missing Librarian

If there is one flaw in the Mac C/MDS environment, however, it is the lack of a librarian program. There is no way to selectively include routines from the libraries short of editing and re-assembling the supplied source code.

Normally the lack of a librarian would be a severe handicap - it still is an inconvenience - but the nature of programming on the Macintosh makes it less debilitating. Since a programmer's primary "library" of routines is in the Macintosh ROM, I think there's less need for the support of an external library.

I use the very smallest C library, and ignore the standard C i/o routines. Since the ROM supports almost all the kinds of i/o and utility functions found in the C library (and then some), I haven't felt deprived at all. I consider the Macintosh ROM a highly specialized library of routines tailored to the Macintosh. I'd no more want to use putchar() on the Mac, than I'd try to draw windows on a TeleType. I guess you could say I've gone native!

Where's the Sieve??

I haven't said anything about performance so far. Where's the benchmark, you may ask. Forget it. Mac C does a highly creditable job of turning your C syntax into assembler. If you need more optimization, the ASM code is available, fully commented, for you to touch-up. A silly little thing like a benchmark won't do justice to the compiler at all, so no matter what you say, I'm not going to stoop to including benchmark results.

Oh, all right, Mac C performs 10 iterations of the Sieve of Eratosthenes benchmark in 4.717 seconds. Satisfied?

If there's one complaint, it's that the compile - assemble - link cycle is so slow. It **is** appreciably slower than most of the other Macintosh C compilers, with the exception of Software Toolworks', but it's not too bad, really, for a microcomputer compiler. With an assist from a hard disk or a RAM disk it's really quite tolerable. A four hundred line program takes nearly four minutes to compile on a floppy disk system, about half that when the source files and linker reside in a RAM disk.

So You C

Mac C brings the power and simplicity of the C language to the Macintosh. I think there's no question C is perfectly suited to programming on the Macintosh, and Mac C is, for my money, the best of the C compilers around. I highly recommend it.

Mac C Compiler
Consulair Corp.
140 Campo Drive
Portola Valley, CA 94025
(415)322-2757

Mac C
Mac C Toolkit
Mac C Compiler and Toolkit
Examples Disk
Floating Point Upgrade

The Electronic Magazine for Macintosh™ Developers

Issue #3

Part II

(Second Release)

7/5/85

How to Print

Harry R. Chesley

Introduction

Most Macintosh applications include a print command. This article tells you everything you need to know in order to implement that command — except, of course, application-specific information about what it is that you're printing.

Printing on the Macintosh is quite straight-forward, whether it's to a local ImageWriter, or a LaserWriter across the AppleTalk network. But it is a good idea to know what's involved, and to keep it in mind when you initially design your application code. This can make the difference between printing being a trivial extension, or a difficult and memory-consuming addition.

How the Macintosh Prints

In the process of using the Macintosh, you've been told that — except in draft mode — it "spools" the information to disk, and then prints it from disk to printer. It's time to understand this process more completely.

For each page in the document being printed, the program issues QuickDraw calls to draw the contents of the page. It is these QuickDraw calls, in the form of a Picture (see Inside Macintosh for more details on Pictures), grouped by page, that are actually spooled to disk.

Next, the Printing Manager allocates as large a block of memory as it can. It uses this block to store a bit-map image of what will go to the printer. This bit-map corresponds to a band as wide as the paper and as long as the available memory allows.

Now the Pictures, one per page, can be read back from disk and executed to fill in the bit-map. Each Picture is interpreted several times, once for each band. The larger the memory block available, the larger the band, and therefore the fewer bands per page. For this reason, it is important that the largest amount of memory possible be available. Therefore, applications generally free up as many non-

essential resources as possible before initiating the second part of the printing process.

Note: This explains a couple of pieces of curious behavior on the part of the Macintosh while printing: (1) It sometimes pauses between bands while printing, but then prints nothing but white-space. The reason is that it must replay the entire Picture for that page to be sure that no drawing within it falls inside the band being processed. If the Picture is complex, this can take time, even though in the end nothing is printed in that band. (2) After printing, applications often access the disk at places that they previously didn't need to. This is because they have purged unnecessary resources from memory — such as menus and dialogs — and must reload them from disk.

Overview of Printing

In order to print, the following steps must be followed:

- Open the Printing Manager.
- For each document to be printed:
    - Tell the manager to start a new document.
    - For each page in the document:
        - Tell the manager to start a new page.
        - Issue QuickDraw calls to draw the page.
        - Tell the manager to end the page.
    - Tell the manager to end the document — a spool file now exists.
    - Free as many resources as possible.
    - Have the manager print the spool file.
- Close the Printing Manager.

In addition to this, there are two dialogs which allow the user to specify how he wants the document to appear. These are called the "style" and "job" dialogs. The style dialog is generally invoked from the "Page Setup..." entry in the "File" menu; and the information supplied is kept with the document and preserved with the document on disk.

The job dialog, on the other hand, specifies the printing resolution, page range to print, etc., and is generally presented to the user each time he selects the "Print..." command. The information from this dialog is not preserved.

Things to do Before and After Printing

Before a document is printed, there are several things which need to be set up or initialized; similarly, there are things which need to be cleaned up afterward.

First, of course, the Printing Manager must be opened at the beginning and closed at the end. This is done with the following two calls:

```
PrOpen;          Initialize the Printing Manager and open the driver.
    .
    .
    .
PrClose;         Close the Printing Manager.
```

The function PrError can be called to determine if an error occurred during these routines, such as not being able to find the printer driver file, or running out of memory.

Second, the application must allocate a print record, which holds — among other things — the information returned from the two dialogs. The print record contents can be taken from the document being processed, or it can be filled in by calling PrintDefault(*print_record_handle*). If it was taken from the document, it should be validated before being used; PrValidate(*print_record_handle*) will return FALSE if everything was in order, or it will correct the record as appropriate and return TRUE.

Third, whenever the user selects the "Page Setup..." entry in the "File" menu, the application should call PrStlDialog(*print_record_handle*). This function will present the user with the style dialog and record his answers in the print record. If the user makes changes and presses the "OK" button, PrStlDialog returns TRUE, and the print record should therefore be changed in the disk copy of the document. If the user presses "Cancel", PrStlDialog returns FALSE, and the print record is not changed.

Once all of this has been done, the application can determine the size of the page with this printer and this style of printing. The print record field prInfo.rPage is a rectangle (type Rect) giving the size of the page. The fields prInfo.iVRes and prInfo.iHRes give the vertical and horizontal resolution in dots per inch.

Printing a Document

The first thing to be done when printing a document is to call PrJobDialog(*print_record_handle*). This will present the user with the job dialog. If the users presses the "OK" button, PrJobDialog will return TRUE; otherwise, it will return FALSE, and the printing should be skipped.

At this point, the application has a completely filled out print record. Printing is now a matter of writing a spool file and then having the Printing Manager print the spool file.

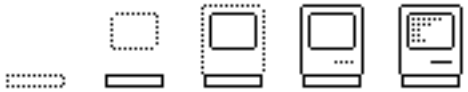
Creating the spool file is a matter of drawing each page in the document, but the page drawings must be preceded and followed by:

```
VAR pPrPort: TPPrPort;
```

```
pPrPort := PrOpenDoc(print_record_handle,NIL,NIL);
      .
      .
      .
PrCloseDoc(pPrPort);
```

The two NIL parameters to PrOpenDoc tell it to allocate the printing port, a pointer to which is returned by the function, and the I/O buffer it will use. A printing port is essentially a GrafPort, but also contains some printing-specific information. PrOpenDoc also makes the port the current QuickDraw GrafPort.

The PrCloseDoc routine frees the storage allocated by PrOpenDoc.



The next step is to draw each page in the document. Surrounding each drawing, the application should make the following calls:

```
PrOpenPage(pPrPort,pPageFrame);
        .
        .
        .
PrClosePage(pPrPort);
```

The pPageFrame parameter to PrOpenPage is a rectangle to be used as a frame for the Picture being spooled to disk. This frame, and the Picture itself, will be scaled to coincide with the page rectangle (see above) when printed. Virtually all applications will want to pass the page rectangle as the pPageFrame parameter so that no scaling is performed.

Now standard QuickDraw calls should be made by the application to draw each page.

There are some limitations when using the LaserWriter. No scaling is done due to the Picture frame — in fact, the LaserWriter driver does not use Picture spooling as described above. And the following limitations apply to QuickDraw calls (this is quoted from Inside LaserWriter):

- Only SrcCopy transfer mode is supported, the other 15 are not.
- The grafverb "invert" is not supported.
- Regions are not supported, try to simulate them with polygons.
- Clip regions should be limited to rectangles.
- Rotated and scaled bit images will not print correctly.
- There is a small error in character widths between screen and printer fonts, so don't rely on them being exactly the same. Only the end points will be accurate. If you are in left, right or center justify mode, only those points will be accurate.

Once the spool file has been created as described above, it can be printed using the following call:

```
VAR prStatus: TPrStatus;

PrPicFile(print_record_handle,NIL,NIL,NIL,prStatus);
```

Note: This call should not be made if draft printing is being performed. Draft printing is done during the first part of the printing process, since it does not use spooling. See the section below on draft printing for more details.

The three NILs tell the routine to allocate a printing port, an I/O buffer, and a band buffer.

Within the print record, the field prJob.pIdleProc is a pointer to a procedure to be executed repeatedly while printing is going on. The prStatus parameter of

PrPicFile is a status record which can be used by this background procedure to tell what the state of the printing process is at that point. If this field is not changed by the application, a default procedure is used which simply checks for the user entering command-period; if it detects this key combination, it halts the printing.

When PrPicFile completes, it releases any storage that it allocated.

Status Boxes

Applications generally display status boxes to let the user know what stage of printing the application is in. These range from the initial "Now saving printed copy to disk." and "Now printing. To cancel, hold down the command key and type a period (.).", to the more sophisticated "Stop/Pause/Continue" buttons of later MacWrite releases.

The following resources can be used to display the first of the above status box sets:

```
* Saving to disk box:
TYPE DLOG
,256
x
100 50 150 450
Visible NoGoAway
1
0
256

TYPE DITL
,257
1

staticText
10 10 45 385
Now saving printed copy to disk.

* Now printing box:
TYPE DLOG
,257
x
100 50 150 450
Visible NoGoAway
1
0
257

TYPE DITL
,257
```

1

staticText
10 10 45 385
Now printing. To cancel, hold down the \11 key and type a period (.).

These dialogs should be invoked by the following code:

```
VAR statusDialog: DialogPtr;

statusDialog := GetNewDialog(256,NIL,POINTER(-1));
DrawDialog(statusDialog);
    .
    .    Do spooling to disk.
    .
CloseDialog(statusDialog);
statusDialog := GetNewDialog(257,NIL,POINTER(-1));
DrawDialog(statusDialog);
    .
    .    Print spool file.
    .
CloseDialog(statusDialog);
```

Draft Printing

Draft printing is different than standard or high quality printing in several ways. Draft printing attempts to use the native character set and capabilities of the printer rather than doing bit-map printing. It also sends printing information directly to the printer rather than spooling it to disk. All of this has several consequences.

The native character set will not exactly match the character set used in the Macintosh. Therefore, the appearance of the printed page will differ from that presented on the screen. Also, graphics will be much more limited, and maybe be omitted entirely.

Since the information is sent directly, if drawing is done in an order other than left to right, top to bottom, the printer may not be able to reverse the direction of printing on the paper in order to match the drawing being done. This can cause loss of information. One of the most common situations where this occurs is when the page has a header and/or footer, which are drawn either before or after the content of the page.

Draft printing has numerous drawbacks. Its one advantage is that it can be much faster than bit-map printing.

Summary

We have described how to print a document from a Macintosh application.

Printing on the Macintosh is very straight-forward, given that the application is designed with it in mind. By this, we mean that the drawing code should be capable

of using the printer port rather than the screen, and that the drawing should be done with a subset of QuickDraw which works with both the ImageWriter and the LaserWriter. It is worthwhile, therefore, to understand how printing works before the application is designed, rather than trying to tack it on afterwards. If it is understood before hand, printing can easily be added to most applications with little effort. If it isn't, printing can be an expensive addition.

The code skeletons provided above should be sufficiently complete to allow a Macintosh application programmer to incorporate printing in his/her application.

Outside Outside Macintosh

The following articles are reprinted from **Outside Macintosh**, Apple's Newsletter for Certified Developers, with permission from Apple.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

On the Road to Publishing Your Software

Writing software for the Macintosh™ is no trivial task. For many Macintosh developers, the stages of software development — after debugging and freezing the code — make up a long and arduous journey. Getting the product to its final destination, the Macintosh users, requires resources and expertise in marketing and distribution.

This article suggests an alternative route for developers who want to write a Macintosh software application, but don't have the resources or the expertise to promote, market, and distribute the product.

The Marketing Challenge

With more than 500 Macintosh products on the market, developers find it more and more of a challenge to gain the Macintosh owners' mind share and penetrate the Macintosh software market. Gone are the days when the key to success was simply to develop a quality product. Today, competition exists not only in terms of quality, but also in innovative marketing and distribution programs.

This competition reaches a peak when major software publishers, such as Microsoft and Lotus, invest in million-dollar advertising campaigns to promote their software. Or when software distributors create irresistible incentive programs for dealers to sell their software. Given these circumstances, even a developer with an innovative and high-quality product will have to reach out further and further to persuade Macintosh users to buy that product.

Great software stems from great ideas. Many developers have realized after many hours of labor, that their idea and product have turned into a costly and time-consuming marketing problem.

Writing and producing a user manual can be more complex than anticipated; the costs of graphic design, artwork, and layout for the manual are likely to run over budget. Getting dealers to display a product on their shelves might be harder than imagined. And another developer may introduce a similar product that is surprisingly able to garner more publicity and more shelf space in dealerships. Indeed, the road to the successful marketing and distribution is full of potholes.

In Search of Publishers

Fortunately, there are many software publishers who can help you market and distribute your products. Just as you're the expert in writing the software, these publishers should be viewed as the experts in software marketing and distribution.

Publishers come in all sizes and names. Assimilation, CBS, Forethought, Hayden Software, and Microsoft are a few examples. For those who aren't sure how to go about finding a publisher, there are even "software agents," such as Zofcom, that specialize in matching Macintosh developers with potential publishers.

Of course, you shouldn't rush into a contract with a publisher. You must weigh the pros and cons of allowing someone else to market and distribute your product. Here are some issues you might want to consider before signing a contract with a publisher:

- **Marketing.** Find out whether the publisher understands the market your software will penetrate. Is the publisher as excited about marketing your product as you are? Check out the publisher's track record: Has he or she published and marketed other software packages successfully? How much control will you, as the software author, have in the marketing process?

- **Advertising and Promotion.** Be sure that the publisher will commit the dollars and other resources needed to successfully advertise and promote your software. Ask for a complete product introduction plan, including advertising, PR, and merchandizing plans. In which trade shows will the publisher be promoting your product? Are you satisfied with the advertising programs planned for promoting your product? Aside from national advertising, is the publisher planning on regional, local, or vertical/trade journal advertising? What promotional and merchandizing tools will be created to support your product?

- **Service and Support.** Be sure the publisher can provide you with technical support while you're developing your program. Is he or she experienced in writing and producing high-quality user manuals? Another important and often-overlooked support area is training — make sure the publisher can provide product training to both dealers and end users.

- **Distribution.** Make certain the publisher is capable of distributing your product through all channels: retail, direct sales, software houses, university/education markets, vertical markets, and direct mail.

Your product will be successful only if developer and publisher can work well together. Before signing a contract with a publisher, your must feel satisfied with the financial and legal terms. And it's equally important for the publisher to be careful when choosing to team up with a developer.

The following interviews with Forethought, Hayden Software, and Microsoft illustrate how publishers typically work with software authors.

Forethought: Great Products, Great Authors

In 1983, Forethought was incorporated by two Apple veterans, Rob Campbell and Taylor Pohlman, for the purpose of developing and publishing software for state-of-the-art microcomputers. They fell in love with the Macintosh and subsequently created the MACWARE line of Macintosh software.

Forethought President Rob Campbell states, "We are committed to invest as much creative energy in the documentation, design, packaging, positioning, distribution, and support of our products as our developers do in creating them."

The MACWARE line now includes FactFinder, a freeform filing system, and Typing Intrigue, a typing tutor combining an individually tailored typing course with two exciting games. FileMaker, a visually oriented database, will be available later this month. Several AppleTalk™ and LaserWriter applications are planned for future release.

The emphasis of the MACWARE line is on personal productivity enhancements and business software, not games or classroom educational/instructional software.

Forethought doesn't do work for hire — they are on the lookout for great products and great authors. Each author must be obsessed with his or her product and must demonstrate imagination and creativity in the use of the Macintosh. Additionally, Forethought wants proof — such as a working prototype — that such a product can be implemented. Campbell feels that if an author and product can meet these requirements, then a successful Macintosh product can be published. This company handles every aspect of producing the software package — from helping standardize the user interface, to writing the manual, to testing, marketing, and distributing the software.

Hayden Software: Year of the Macintosh

Since spring 1984, Hayden Software has committed to investing most of its development resources in the Macintosh. To date, Hayden has published more than 15 Macintosh packages for the professional and home markets, including the popular MusicWorks, VideoWorks, Sargon III, Ensemble, and the DaVinci series. To stimulate sales of all their Macintosh titles, Hayden has embarked on an ambitious advertising and marketing campaign called 198-MAC: Hayden Software's Year of the Macintosh. 198-MAC, which started last October, is a year-long campaign that includes the introduction of more than 20 Macintosh titles to be published by Hayden.

Other elements of the campaign are MacTube, a dealer incentive program for selling Hayden Macintosh software, and two other dealer programs — MacRac and MacSweep.

The MacRac campaign, conducted earlier this year, provided every participating

computer dealer a free rack for displaying Hayden Macintosh software. Dealers who used the MacRac in their stores qualified for four co-op ads: two in the *Wall Street Journal*, one in *Personal Computing*, and one in *MacWorld*.

Probably the most ambitious component of 198-MAC is MacSweep, an inventory buy-back program for software retailers. In this "spring house-keeping" promotion, for every $3 of Hayden software bought, a dealer can return $1 of "dead software in inventory" from any publisher on any machine.

Accompanying these marketing programs is a heavy advertising campaign. The whole line of Hayden Macintosh products will be advertised on a national level, such as in *Personal Computing* and *MacWorld;* specialty software such as Sargon III will be advertised in such vertical market magazines as *Chess Life*.

Microsoft: Check Out MacLibrary

Microsoft, known mainly for its languages, operating systems, and general applications, has set up MacLibrary — a publishing house for non-Microsoft writers.

Alan Boyd, MacLibrary's publisher, explains that Microsoft's visibility, marketing expertise, distribution network, and experience in producing software bring the software writer all the ingredients needed to make a product successful. In return, the software writer provides software that is usually too specialized to be cost-effective for Microsoft to write in-house.

Its experience with Flight Simulator and Project, both written by third-party authors, has enabled Microsoft to develop a smooth system for working with software authors. Each author is assigned a technical adviser, a managing editor, and a project manager. Working with these three people, the author is able to produce a quality software package while letting Microsoft handle the burden of marketing and distribution.

Microsoft is looking for developers with working prototypes in most Macintosh application areas: productivity tools, programming tools, university-level education, self-improvement, advanced recreation, and innovative Macintosh-inspired software.

Three MacLibrary products are currently available: Logo, Entrepeneur, and Learning Microsoft Multiplan and Microsoft Chart. Six additional products are currently in the development process.

Products in MacLibrary are developed with a consistent MacLibrary look and feel. MacLibrary packages are marketed together on an international level and share the same distribution network as other Microsoft products.

An Author's Success Story

Mark Cantor is the author of two successful packages, MusicWorks and VideoWorks, both published by Hayden Software. Reporting that he has enjoyed working with Hayden, Cantor lists several of the company's strengths:

- Hayden is very visible; therefore his products are very visible.
- Hayden has been responsible for the production, marketing, and

distribution of his products, while allowing him a say in these matters.

- Hayden is aggressively marketing and promoting his packages. For example, it is bundling VideoWorks with another author's image digitizer.

Cantor believes that before choosing your publisher, it's important to know your product and who will want to buy it. Many of the publishers specialize in certain types of products or markets — for examples, Cantor feels that Assimilation and T/Maker are strong desk accessories publishers. By knowing your product, you then choose the publisher with the resources and visibility to get it to the right people.

Write Your Code and Relax

If you feel you don't have the resources or expertise to produce and market your package, you may want to contact a software publisher. A list of Macintosh software publishers is available through the Macintosh Developers Group.

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

Developer Associations

By Eric Zarakov, Macintosh Evangelist

Developers have started to band together to form organizations known as *developer associations*, which typically meet once a month to socialize, exchange ideas and techniques, discuss problems, and share knowledge. Participation in these nonprofit groups is beneficial both to participating developers and to Apple.

Benefits of Developer Associations

Participating in an association can assist developers in a variety of ways. For example, it can — through agreement by the parties concerned — reduce the risk of duplicate development on similar projects that might be in competition and not sufficiently different to warrant parallel development. Associations might provide a central library for articles, periodicals, development tools, software, and documentation.

Attendance at monthly meetings is a good way to keep apprised of new developments from Apple. Wild stories and rumors (justified or unjustified) can be confirmed or denied when one person from the association is designated as an Apple contact.

Dissemination of technical information has already proven valuable for both Apple and the Austin Area Certified Developer Association. Some of the members of this Texas group experienced some difficulties associated with the February Software Supplement from Apple and the Lisa Office System 3.1. A single phone call from Julio Barrada, Association chairman, prevented Macintosh Technical Support from receiving a flood of phone calls from other Austin developers. Julio was able to

resolve his own problem and relay the information to other Association members. In other words, one call helped everyone! Developers who work together find that their problem solving is expedited by volunteer support from local experts who are association members. The Austin organization, for example, lists four members who have volunteered their time and expertise in various aspects of Macintosh development. Association members are asked to solicit help from these local volunteers only after they have exhausted all other possible resources.

Remember that as of April 1, Macintosh Technical Support is available only to registered developers; certified developers will receive technical support only through electronic mail. This is a good incentive for certified developers to form local developer associations.

I'd like to thank Julio Barrada from the Austin Area Certified Developer Association for pioneering the road to developer associations. Should you want to contact him, Julio can be reached at the following address:

Austin Area Certified Developer Association
P.O. Box 50447
Austin, TX 78763
(512) 441-4583

Clearinghouse Will Help

To facilitate the forming of associations, the evangelist group will become a clearinghouse of some information. For one thing, we'd like to provide a list of developer associations. If you want to be on our list (existing associations only), please send me the name of your association. To obtain a list of current associations, please write to me requesting the list; write Attention: Developer Association List on the envelope. Send your information and requests to:

Eric Zarakov, Macintosh Evangelist
Apple Computer M/S 4T
10443 Bandley Drive
Cupertino, CA 95014

If you want to locate other developers in your area who are interested in forming and association, put a classified ad in *Outside Macintosh*...it's free!

◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊