THE ELECTRONIC MANUSCRIPT

Moving from hard copy to electronic copy requires a clear understanding of technology

■ By Galen Gruman

So, after reading success stories about publications that have used desktop publishing or electronic typesetting at great cost savings with no loss in quality, you're ready to take the plunge. You've bought Xerox Ventura Publisher, Aldus PageMaker, or QuarkXPress. Or perhaps a local typesetter is advertising huge discounts if you send your precoded type via modem. You see an opportunity to reduce your typesetting costs from $40 or $50 a page (through traditional methods) to $8 (through desktop publishing) or $23 (via electronic typesetting).

But wait! You don't generate a lot of your copy in-house, so everything you get must be rekeyed or translated from the various word processors and computers that your writers use. And having the text rekeyed costs a lot: either secretarial time (and salary) or editorial time (and salary). In traditional publishing, this cost was hidden as part of the typesetting charges.

There are two basic barriers to file transportability: data format and medium format. Data format is the structure used by a particular program; medium format is the structure used by the disk drives of a particular computer type.

As an example of data formats, consider word-processing packages on an MS-DOS (IBM-compatible) PC. Popular packages include Word Perfect, Microsoft Word, and Wordstar. But they all store their text in different data formats, even though they run on the same MS-DOS machines.

As an example of medium formats, consider MS-DOS and Apple Macintosh microcomputers. They write data on their disks differently, so even though a disk from one might physically fit in the other's drive, it isn't readable; the data is stored in different places than expected. For example, both the MS-DOS and Mac versions of PageMaker store their data in the same format. Because an MS-DOS PC can't read a Mac disk, however, the fact that the data formats for PageMaker are the same on both machines doesn't help you. You have to get around the medium-format incompatibility to take advantage of PageMaker's data-format compatibility.

How do you handle this Babylonian variety of formats? There are several answers, and you'll have to tailor them to your specific environment. But there *are* answers.

Hardware sets the stage

To get involved in desktop or electronic publishing, the choice will come down to an MS-DOS PC or a Macintosh. Both have their advantages and disadvantages. MS-DOS machines outnumber Macs 6 to 1 in the corporate world and are the *de facto* worldwide standard for most personal applications. But the Mac has a strong following among graphic artists.

If you already have some computers, buy more of the same. If you have none, get some hands-on experience

Continued on page 42

Many Mac applications come with an option to save in MS-DOS format—a recognition of MS-DOS's overwhelming popularity.

running the programs you'll be using every day. Don't believe what the salespeople say; do what works best for you after you've tried it for a while. If necessary, rent some machines for a month or talk to colleagues who are using computers.

One important fact to remember is that both types of machines have excellent desktop-publishing programs: Neither is superior to the other in this very important respect. Your decision will probably come down to three points:

- *What you have already and are comfortable with and/or what your contributors have.* Don't change your environment if you're comfortable with it.
- *How oriented you are to graphics versus how oriented you are to text.* Macs are superior in graphics; MS-DOS PCs are superior in text. As both systems evolve, these differences are diminishing.
- *Your budget.* Macs cost two to three times that of an equivalent MS-DOS machine because only one company (Apple) makes and sells Macs. MS-DOS PC add-ons tend to be cheaper than Mac add-ons as well. MS-DOS machines can be expensive if you buy from the biggest names (IBM and Compaq), but many other companies produce machines as reliable (and often with extra features) for much less. The inside components are the same, so don't worry about the label on the outside too much.

If you buy Macs, you'll find that the word-processing programs you buy will generally read files created by the popular Mac word processors, so accepting files in popular Mac data formats won't be that big a deal. Many Mac applications come with an option to save in MS-DOS format—a recognition of MS-DOS's overwhelming popularity. On the flip side, it is unheard of for MS-DOS programs, except for some graphics and desk-

HOW ASCII WORKS

ASCII, the American Standard Code for Information Interchange, is essentially an alphabet for computers. Computers store information as a series of 0's and 1's (zeros and ones called "bits," which comes from the term "binary digits"). The computer's processor reads bits in groups called "bytes," which are eight bits long.

The position of the 0's and 1's determines the byte's value. Values grow from right to left. To translate a string of bits into the decimal numbers that people use, here's what you do: First, write down the values of each position from left to right. If you have eight bits, they are 128, 64, 32, 16, 8, 4, 2, and 1. If the bit is 1, you add the value for the position; if it is 0, you don't. For example, the byte "10000001" equals 129, because the first bit's value is 128 and the eighth bit's value is 1. Those add up to 129. Because the second through seventh bits' values are 0, you don't add in their values.

With eight bits, a byte can represent 256 different meanings or characters, one for each possible combination of 0's and 1's. Because the leftmost bit is usually used as the equivalent of a space, that leaves seven usable bits, which allows 128 meanings or characters. The ASCII standard defines what character each combination means.

In ASCII, the first 32 bytes (00000000 through 00011111, or 0 through 31 in decimal numbers) are reserved for control characters such as Break, Carriage Return, Line Feed, Form Feed, End of File, End of Take, and Beep. Values 32 through 47 are for punctuation, 48 through 57 are for numerals, 58 through 63 are for more punctuation, values 64 through 90 are for uppercase letters, 91 through 95 for still more punctuation, 96 through 122 for lowercase letters, and 123 through 126 for even still more punctuation, and 127 for the rub-out character. In ASCII—no matter what the computer is—the letter "A" always has the value 65, which the computer sees as the byte 01000001, and the character "[" always has the value 95, which the computer sees as the byte 01011111.

Because ASCII is the universal standard for information on personal computers, it guarantees that anything in ASCII format can be read by any PC, regardless of who made it. (You still have the problem of whether the disk formats are compatible.) ASCII is thus the lowest common denominator for information exchange.

Although your word processor format may be different than that of your freelance writers, you know that if both word processors can read and write ASCII that you can transfer text. But ASCII does not include formatting information, so formatting like justification, italics, and boldface are not retained. If you use ASCII files to transfer text between word processors, between a word processor and a typesetting program, or over a modem, you'll have to embed codes to indicate any formatting information.

ASCII incompatibilities

Unfortunately, the universality of ASCII has been reduced by both Apple and IBM in efforts to *improve* the standard.

APPLE—Apple decided that it was redundant for ASCII to have control characters for both a carriage return and a line feed, so the company decided not to use the line feed in the

GRAPHICS FILES

Like text files, graphics files come in all sorts of formats. Fortunately, most people don't edit graphics files sent in by others, so the translation problem that exists for editors is less acute for layout artists and those who handle graphics. Of course, if you do intend to edit others' graphics files, follow the same advice that I gave for text: Accept only those formats for which you can edit in native mode or that you can easily translate. There are two basic kinds of graphics files: raster and vector.

RASTER—Raster files are made up of a series of black-and-white dots that, like a halftone, simulate a continuous-tone image. These are popularly called paint, picture, and bitmap images. The resolution of such an image is determined by the program that creates it (or by the scanner that you use to read it in), since that program determines how many dots per inch of information are stored. That means an image created at or scanned in 300 dpi will output at no better than 300 dpi—even on a 1,270-dpi or 2,540-dpi typesetter.

Raster files are the easiest to translate, and several programs are available to do it. The higher-end desktop-publishing programs (both Macintosh and MS-DOS) can import most popular formats and display them on screen, so a translation program is rarely needed unless you intend to edit the picture. Color is usually translated to grays or simply black and white, depending on the format. Popular formats include MacPaint, PC Paintbrush (PCX), Graphics Environment Manager/Dr. Halo (IMG), Tagged Image File Format (TIFF), and Windows metafile.

VECTOR—Vector files are made up of instructions on what to draw and how to draw it, such as "Draw hollow circle with 12-inch diameter with one-point border with center point at coordinate 14,5." These are popularly called object-oriented, line-art, and drawing images. The resolution of such an image depends on the output device's resolution, since the file tells the device to draw it at its maximum resolution.

Vector files are more difficult to translate than raster images, so there are fewer translation programs available for them. Higher-end desktop-publishing programs usually import several of these, although any text in them usually converts to Helvetica or Times Roman during the process. Popular formats include Computer Graphics Metafile (CGM), Drawing Interchange Format (DXF), Encapsulated PostScript (EPS), Graphics Environment Manager (GEM), Hewlett-Packard Graphics Language (HPGL), Lotus (PIC), and Pict (an Apple Macintosh format now becoming available in some MS-DOS programs).

NOTES—Most formats are straightforward: Either your program can read them or not. But two formats warrant special attention:

- EPS files can contain bitmap instructions as well as drawing instructions. Because EPS files are stored as plain text, if transferred from a Macintosh to an MS-DOS PC, they can have the same missing line-feed problem that ASCII text files do (see the box on page 42 for details). As with text, you may have to run them through a program to add the missing line-feed character. (Transfer programs on the Apple side often will not do this because EPS files are marked as graphics on a Macintosh and thus as files that the transfer program is told not to adjust. Because they are really text files—no matter how the Mac marks them—they can be adjusted with no danger. Newer transfer programs recognize and properly adjust EPS files during transfer.)

  Ventura Publisher can create a pseudo-EPS file that is handy for outputting a full page and then using it as a piece of artwork (such as for a cover repeat or a resized logo). Although this pseudo-EPS file will print on any PostScript printer and can be imported into any Ventura document, no other program that reads EPS files can import it.

- The Pict format is evolving, so artwork from programs like Apple's MacDraw II and Micrografx's Designer may not import correctly or at all. For example, Ventura Publisher reads MacDraw I Pict files correctly, but MacDraw II Pict files with all objects containing text are translated to solid black, and Designer Pict files will not import at all. Apple, Xerox, Micrografx, and others, dispute whose responsibility it is to handle this incompatibility. Claris, a software company spun off from Apple whose products include MacDraw II, calls the Pict in MacDraw II "Pict 2," emphasizing the difference between original Pict and the new version. This differentiation is certain to cause headaches for both users and developers of programs that import Pict graphics.

top-publishing programs, to read Mac data formats. There are also disk drives available for Macs that can read and write the MS-DOS disk (medium) format.

Of course, if most of your contributors use PCs, you'll find yourself spending a lot of money making your Macs able to handle MS-DOS. In that case, why not just use MS-DOS PCs?

The determining factor may come down to what software you like best. If you like QuarkXPress the best for desktop publishing and Apple MacWrite for word processing, you'll have to buy a Mac. If you prefer Xerox Ventura Publisher and Xyquest Xywrite III, an MS-DOS PC is the way to go.

Whatever machine you get, a modem will be invaluable. It lets you connect microcomputers to each other (even those of different types), to mainframe computers, and to electronic bulletin boards and electronic mail services. A modem, which translates computer coding into sound pulses and then back, uses a phone line to connect remote or otherwise unconnectable computers. You can use it to let your contributors dial in to your PC and transfer their files to you or to connect a Mac to a MS-DOS PC to move files between them.

A modem works easily with files in ASCII data format (see below for an explanation of "How ASCII Works"), an American standard for data that is used throughout the world on PCs and many other computers. Almost every program—spreadsheets, word processors, and database managers—can output in ASCII format (printing to file is the most common way). Many can read in ASCII format, too. The disadvantage of ASCII is that it only works for plain text, so any formatting is lost. And nontextual data, like graphics and the calculations in a spreadsheet, can't be sent in ASCII format.

Continued on page 44

Macintosh. In Apple, the carriage return automatically adds a line feed, so Apple programs insert only the carriage-return code when you press Enter. But in the IBM world, computers insert both characters whenever you press the Return or Enter key. If you are moving files from IBM machines to Apples, there's no problem, but there is coming back the other way.

Why are there both characters in ASCII? Because teletypewriter coding—from which ASCII was derived—needed both. If you wanted to underline something on a teletypewriter, you issued a carriage return to go back to the beginning of the line and then spaced over to where you wanted to type the underlines. You used the same process for boldface. If you had a bidirectional printer, you would issue the line-feed command to move down to the new line and then enter the line's characters in the reverse direction of the previous line. If you wanted to move to the beginning of a new line, you needed both.

Most newer Macintosh programs have an option to add in the line feed for users who intend to send the files to an IBM-compatible PC. Many of the transfer programs, however, strip the line feed right back out. I've had difficulties even with transfer programs that give you the option of retaining the line feed, so I wrote a program to put it back in once the file was on my PC (see the third program in the box on page 46).

IBM—IBM decided that the unused space bit was the perfect way to double the number of characters while still sticking to the ASCII standard, so IBM assigned foreign letters, special symbols, and line-drawing characters to values 128 through 255, which the company called "extended ASCII." All IBM had to do was make the first bit 1 instead of the standard 0; making that first bit 1 adds 128 to the byte's value.

Unfortunately, non-IBM-compatible machines won't understand these extended ASCII characters, and neither will modems. What most will do is ignore the first bit to get a legal ASCII value (one whose first bit is 0). Because the value of the first position is 128, ignoring that first bit is like subtracting 128 from the byte's value if that value started with 1. For example, if you use value 224 ("11100000" in binary) to get the extended ASCII "[α]" symbol (Greek lowercase alpha) and then modem the file to someone else, he will get value 96 ("01100000" in binary), which is "`" (the open single quote).

And even on IBM machines, not everyone assigned the same characters to the extended ASCII values that IBM did. For example, neither Ventura Publisher nor PageMaker interprets the extended ASCII characters the same as IBM or as each other, although there is some overlap among all three.

The International Standards Organization is working on a new ASCII that standardizes the use of the values 128 through 255. The emphasis is on using those values for European accented letters and European punctuation. The group is also considering an ASCII that would handle Japanese and other Oriental languages as well. Such an ASCII would require 16 bits to handle the larger number of characters. Meanwhile, it's safest not to use the extended ASCII sets if you're using a modem or transferring files to people using different programs than you are.

SIMPLE ASCII TRANSLATION PROGRAMS

Here are three programs written in Basic to take ASCII files transmitted by modem or from a Macintosh. They will run under the MS-DOS Basic interpreter (the Basica or GW-Basic interpreters). If you are a Basic programmer, you can rewrite these to work faster and to have a custom interface. Be sure to compile these programs if you have a Basic compiler—the speed will increase as much as 700 percent.

1. Modemed ASCII file with two returns signifying a new paragraph. Paragraph indents are translated to the tab character (the Basic instruction "CHR$(9)"). This program strips out any extended ASCII characters; delete line 130 if you want to retain them.

```
10 CLS : PRINT "Translate ASCII files with 2 returns indicating new paragraphs" : PRINT

20 INPUT "File name to translate (include disk and extension): ",A$

30 IF MID$(A$,2,1) <> ":" OR MID$(A$,(LEN(A$)-3),1) <> "." THEN PRINT CHR$(7);
   ">> Bad file name." : GOTO 20

40 IF RIGHT$(A$,3) = "ASC" OR RIGHT$(A$,3) = "asc" THEN PRINT CHR$(7);">>
   The .ASC extension is reserved for the translated file." : PRINT " Rename the source
   file and rerun the program." : PRINT : END

50 C$ = LEFT$(A$,LEN(A$)-3) + "ASC"

60 PRINT : PRINT "Translating ";A$;" ..." : PRINT

70 OPEN A$ FOR INPUT AS #1 : OPEN C$ FOR OUTPUT AS #2

80 IF EOF(1) THEN CLOSE : GOTO 210

90 B$ = INPUT$(1, #1)

100 Z$ = B$

110 IF B$ = CHR$(10) THEN RNFLAG% = RNFLAG% + 1 : GOTO 190

120 IF B$ > CHR$(0) AND B$ < CHR$(32) THEN GOTO 190

130 IF B$ > CHR$(127) THEN GOTO 190

140 IF RNFLAG% > 2 THEN Z$ = CHR$(13) + CHR$(10) + CHR$(13) + CHR$(10) + Z$

150 IF RNFLAG% = 2 THEN Z$ = CHR$(13) + CHR$(10) + CHR$(9) + Z$

160 IF RNFLAG% = 1 THEN Z$ = " " + Z$

170 RNFLAG% = 0

180 PRINT #2, Z$;

190 Z$ = ""

200 GOTO 80

210 PRINT : PRINT CHR$(7);">> Translation complete." : PRINT " ";C$;" contains the
   translated text." : PRINT

220 END
```

Continued on next page

When you send an ASCII file over a modem, you usually have to send it in line mode, which means (1) that you must have a carriage return (called a hard return in computer parlance to distinguish it from the soft return of a word processor's line wrap) at the end of each line and (2) that each line must not exceed 80 characters. That means you can't differentiate line breaks from paragraph breaks unless you use a convention like putting two returns or a return and five spaces to indicate a paragraph break. Then you have to manually strip out all the extra returns before editing it in your word processor. You can write programs to do this stripping automatically. ("Simple ASCII Translation Programs" at the left provides two examples.)

Modems can also transmit non-ASCII (usually called binary) data formats, but only if both modems are equipped with the same transfer protocol. These protocols include Kermit and Xmodem, which can be used to send non-ASCII formats (like your word processor's native format) between computers. These are handy when sending files in a format available on both a Macintosh and an MS-DOS PC, like PageMaker or Microsoft Word (the Apple version has an option to read and write in the MS-DOS format).

Translation software

If you have contributors using several word processors, you can buy a translation program (there are dozens available) to convert among several popular formats. Your word processor may have even come with a built-in or separate converter to import (read in) other formats into it. The problem with translation software is that formats often change as programs get enhanced, so you may have to get new versions of your translator as you or your contributors trade their word processors for enhanced versions. In addition, not all formatting translates among all word processors, so you'll have to experiment beforehand so you can anticipate what might get lost in the translation.

Desktop-publishing programs also can import a variety of formats—Ven-

tura Publisher can export (output) in eight and PageMaker in two—so you might use your desktop-publishing program as a translator.

If you work in a mixed-computer environment, you'll want a transfer/translation program to move files between Macs and MS-DOS PCs. These programs can do nothing but connect a Mac to an MS-DOS PC like a modem would so you can exchange ASCII files; or they can translate data formats as they move the data from one machine to another. Some of the smarter ones even handle the slight differences between MS-DOS and Mac ASCII (see "How ASCII Works" on page 42). Most are hardware/software combinations: The hardware connects the machines physically; the software translates the data formats and manages the transfer process. They usually use your modems or a null cable (a cable connected to the serial ports of the two computers).

For occasional data- or medium-format translation, you might consider using a translation service. Many cities now have companies that transfer data among formats. These can get expensive, however, so use them only for exceptional needs. Get your own software and equipment if a translation need recurs frequently.

Setting standards

The best way to handle receiving electronic copy is to set some house standards. After some grumbling, you'll be amazed how quickly contributors and staff adapt. Such standards, after all, are no different than expecting people to follow house editorial style or submission guidelines for length and margins.

The first thing to do is settle on a house word processor. Try to pick one that directly imports ASCII files as well as its native format. Then tell your contributors they may send you files in the word processor's format or in ASCII. That'll remove your translation headaches (perhaps by offloading them to your contributors). You might also let contributors send their text in one or two other formats for which you have a dependable translator. The key, however, is getting something that

Continued on page 46

SIMPLE ASCII TRANSLATION PROGRAMS

Continued from page 44

2. Modemed ASCII file with one return followed by five spaces signifying a new paragraph. Paragraph indents are translated to the tab character (the Basic instruction "CHR$(9)"). This program strips out any extended ASCII characters; delete line 140 if you want to retain them.

```
10 CLS : PRINT "Translate ASCII files with 5 spaces indicating new paragraphs" : PRINT : PRINT
20 INPUT "File name to translate (include disk and extension): ",A$
30 IF MID$(A$,2,1) <> ":" OR MID$(A$,(LEN(A$)-3),1) <> "." THEN PRINT
   CHR$(7);" >> Bad file name." : GOTO 20
40 IF RIGHT$(A$,3) = "ASC" OR RIGHT$(A$,3) = "asc" THEN PRINT CHR$(7);" >> The
   .ASC extension is reserved for the translated file." : PRINT " Rename the source file and
   rerun the program." : PRINT : END
50 C$ = LEFT$(A$,LEN(A$)-3) + "ASC"
60 PRINT : PRINT "Translating ";A$;" ..." : PRINT
70 OPEN A$ FOR INPUT AS #1 : OPEN C$ FOR OUTPUT AS #2
80 IF EOF(1) THEN CLOSE : GOTO 240
90 B$ = INPUT$(1, #1)
100 Z$ = B$
110 IF B$ = CHR$(10) THEN RNFLAG% = RNFLAG% + 1 : GOTO 220
120 IF B$ = CHR$(32) THEN SPFLAG% = SPFLAG% + 1 : GOTO 220
130 IF B$ > CHR$(0) AND B$ < CHR$(32) THEN GOTO 220
140 IF B$ > CHR$(127) THEN GOTO 220
150 IF RNFLAG% = 1 AND SPFLAG% = 5 THEN Z$ = CHR$(13) + CHR$(10) + CHR$(9)
   + Z$ : GOTO 200
160 IF RNFLAG% = 1 AND SPFLAG% > 0 AND SPFLAG% <> 5 THEN Z$ = CHR$(13)
   + CHR$(10) + Z$ : GOTO 200
170 IF RNFLAG% > 1 AND SPFLAG% = 5 THEN Z$ = CHR$(13) + CHR$(10) + CHR$(13)
   + CHR$(10) + CHR$(9) + Z$ : GOTO 200
180 IF SPFLAG% > 0 THEN Z$ = SPACE$(SPFLAG%) + Z$
190 IF RNFLAG% > 1 THEN Z$ = CHR$(13) + CHR$(10) + CHR$(13) + CHR$(10) + Z$
200 IF RNFLAG% = 1 THEN Z$ = " " + Z$
210 RNFLAG% = 0 : SPFLAG% = 0
220 PRINT #2, Z$;
230 Z$ = ""
240 GOTO 80
250 PRINT : PRINT CHR$(7);" >> Translation complete." : PRINT  ;C$;" contains the translated
   text." : PRINT
260 END
```

Continued on page 46

When you create a markup system, make it logical, short, consistent, understandable, flexible, and extendable.

SIMPLE ASCII TRANSLATION PROGRAMS

Continued from page 45

3. ASCII file from a Macintosh that lacks the line feeds that MS-DOS needs to recognize a carriage return. Some programs that transfer files between MS-DOS and Macintosh PCs now automatically add the missing line feeds, as do some Macintosh programs that have Save As options for MS-DOS format.

```
10 CLS
20 INPUT "File name to translate (include disk and extension): ",A$
30 IF MID$(A$,2,1) <> ":" OR MID$(A$,(LEN(A$)-2,1) <> "." THEN PRINT
   CHR$(7);">> Bad file name." : GOTO 20
40 IF RIGHT$(A$,3) = "ASC" OR RIGHT$(A$,3) = "asc" THEN PRINT CHR$(7);">>
   The .ASC extension is reserved for the translated file." : PRINT "Rename the source file
   and rerun the program." : PRINT : END
50 C$ = LEFT$(A$,LEN$(A$)-2) + "ASC"
60 OPEN A$ FOR INPUT AS #1
70 OPEN C$ FOR OUTPUT AS #2
80 IF EOF(1) THEN CLOSE : GOTO 140
90 B$ = INPUT$(1, #1)
100 Z$ = B$
110 IF B$ = CHR$(10) THEN Z$ = CHR$(13) + CHR$(10)
120 PRINT #2, Z$;
130 GOTO 80
140 PRINT : PRINT CHR$(7);">> Translation complete." : PRINT ;C$; "contains the
   translated text." : PRINT
150 END
```

you can work with in your word processor's native format.

Next, tell your contributors what media you can accept—what types of floppy disks and for what machines—and how to access you directly (via an electronic mail service like Genie, Compuserve, or MCI Mail, or by dialing directly into one of your modem-equipped computers running a bulletin-board program). If you can accept files through such direct access, you'll find that you save a lot of turnaround time—and express-delivery charges. After all, a phone call is instant. Direct file transmission is a lot like fax: once you use it, you're hooked.

Last, and most important, insist on electronic versions. You should get to the stage within a year where you don't know what to do with hard copy. At magazines I have been involved with, only letters to the editor and the occasional article from a far-off place like China had to be rekeyed after the move to electronic publishing.

Marking up on-line copy

Once you receive copy electronically, you'll find that a lot of the markup you did on paper has no counterpart in your word processor. And it certainly has no counterpart in ASCII. Unfortunately, there's no widely accepted markup convention like the Chicago symbols used by proofreaders. (The University of Chicago did come out with a guidebook for electronic markup, but it is poorly designed and aimed mostly at book publishers.) You'll have to invent your own or use your desktop-publishing or typesetting codes.

If you have a knowledgeable programmer on hand, even one whose only language is Basic, I recommend that you come up with an independent set of codes and have that programmer write a program that translates the codes into the desktop-publishing or typesetting codes. If you don't have such resources, you might still come up with such an independent set of codes and then use your word processor to search and replace those codes with the final codes after you're done editing.

Why? If you ever change packages or typesetters, you don't have to relearn codes. Also, most typesetting codes work differently than the markup codes editors are used to. (Consider spec'ing bold italics with markup versus typesetting codes. With markup, you use a straight line and a wiggly line under the bold italic type—an additive process. In typesetting, you either make an explicit change to the bold italic face for the particular font or you invoke a code that means bold italic. You wouldn't invoke the separate codes for italic and bold because the typesetter will only implement the last face-change command it sees—a literal process.) Also, typesetting codes are not readily readable by authors. I believe strongly in keeping your editorial process independent of your hardware setup.

If you do create your own codes, here are some hints:

- *Use a seldom-used character to invoke them.* (These are called delimiters.) I prefer brackets ("[" and "]") because they aren't used too often in

text and because I don't have to press the shift key to access them. Many programs use the vertical wedge-shaped caret open and close marks, or less-than/greater-than math symbols ("<" ">"), which are less frequently used but harder to type. When you do have to use your delimiters as actual characters, a common practice is to use them twice. For example, if "[" is a delimiter, "[[" means print the "[" character. If anything else followed the first "[" your program (or search and replace) would expect a code.

• *Use fixed-length, short codes.* Many common coding schemes (including the Chicago on-line and, to a lesser extent, the Association of American Publishers's provisional electronic markup standard) use variable-length, long codes that are difficult to type and remember. They're also harder to spot errors in than fixed-length codes because a missing or extra character is more obvious if a code's length is always the same. Such long, variable-length codes are fine to use for automatic markup systems that translate word-processor or shorter code formats into those formats, but if you're not sending type to someone who insists on them, they have little value.

• *Use repeating mnemonics in codes.* For example, you might start all math symbols with "m" or consistently use "o" and "x" at the end of the codes to indicate "on" and "off," respectively.

• *Separate codes by type.* There are four basic types of codes: those for attributes (like boldface and superscripts), those for text types (like headlines and text), those for symbols, and those for typographic controls (like justification).

You can get away with just two characters in most codes, including attribute and common-symbol codes. Examples include "[io]" and "[bx]" for italic on and bold off, respectively. For rare symbols, use a code that means "symbol" (like a font change to a pi font) and then a number that represents the symbol (this could be based on the keyboard layout position for that symbol). An example is "[ss123]" for character 123 of the symbol set. Another is "[ps36]" for 36-point type. You can use the same two-character approach for text-type codes if you use levels. For example, you can use "[bt" to indicate body text and follow it with the appropriate level. For example, "[bt1]," "[bt2]," and "[bt3]" represent first-, second-, and third-order body text. If you are used to talking about first-level heads or second-level text, this numbering scheme will be even easier. It also preserves the two-character code structure by extending the basic codes with numerals indicating levels, sizes, and other values.

When you create a markup system, make it logical, short, consistent, understandable, flexible, and extendable. It can be difficult to meet all these objectives, so you may have to spend some time trying out alternative schemes before you come up with one that meets your needs.

Putting it all together

Once you've established a system and narrowed the variety of formats you'll accept, you'll find that electronic copy is no harder to work with than paper copy. Plus you get all the benefits of on-line editing (changes were never so easy to make or correct!) and of electronic publishing (output costs were never so low!). The key is not to accept text in any format it arrives in. After all, how many editors accept articles typewritten in script or Gothic characters?

With these standards in place, you'll find that the transition from paper to electronic copy will be smoother and faster than if you just leap without looking. Yes, there will be some bumpy times during the transition, but they'll be worth the benefits gained. ■

Galen Gruman reviews desktop-publishing software for the computer industry trade weekly InfoWorld *and produces the Computer Press Association's publications.*