Inter-Office Memorandum

To	Whom it May Concern	Date	July 19, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	Guide to [Ivy]<AltoFonts>	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>AltoFontGuide.bravo
[Maxc1]<Fonts>AltoFontGuide.press
[Ivy]<Fonts>Memos>AltoFontGuide.bravo
[Ivy]<Fonts>Memos>AltoFontGuide.press

In the early part of 1980, I poked around the InterNet and gathered up all of the Alto fonts that I could find. With the assistance of Lindsey Halloran, I edited these font to update them to the latest character code conventions: that is, I put in the third generation of special characters. I then shipped all of the resulting Alto fonts onto the directory [Ivy]<AltoFonts>. Different versions of a particular font are distinguished through the use of subdirectories. The purpose of this memo is to explain what those subdirectories are. If you need an Alto font for, say, TimesRoman10, I recommend that you List all of the files that match the pattern

[Ivy]<AltoFonts>*TimesRoman10.al,

and then choose from among them on the basis of the following information.

Criteria for choosing an Alto font:

One important reason to prefer one Alto font over another is aesthetics: you like it because you like it. I don't presume to make any judgments on that one. But the other two important criteria are unfortunately inversely correlated: fidelity and legibility. If the screen is to be a faithful model of the printed page, then characters on the Alto screen should be the same size as the ones coming off the printer. But, since the Alto display is a low resolution bitmap, it is hard to draw characters that are both small and good-looking; in particular, it is hard to get them both thin and good-looking. Thus, the designers of most Alto fonts establish some compromise between the thin characters that fidelity demands, and the wide characters that are easier to read. Different Alto font artists have assumed different compromise positions over the years; now, you can look over all their results and choose the one that best fits your needs.

Here is a brief guide to the subdirectories:

[Ivy]<AltoFonts>Original>
updated versions of [Maxc1]<Altofonts>
wide, very legible
[Ivy]<AltoFonts>Thin>
Pellar's second generation fonts
rather thin, not very legible
[Ivy]<AltoFonts>RoundedWidths>
updates of some fonts that Pellar did for ASD
thinnest of all: each character has as width the result
of rounding the printing width to the nearest Alto dot.

. . . (continued on next page)

[Ivy]<AltoFonts> (that is, the null subdirectory)
Pellar's third generation fonts, except for codes above #177
roughly the same width as Original, but a different feel
[Ivy]<AltoFonts>EightBit>
Pellar's third generation fonts, complete
roughly the same width as Original, but a different feel

First generation Alto fonts: <AltoFonts>Original>

The first generation Alto fonts are very legible, partially because they are fairly wide. When these Alto fonts were first drawn, there was only one special character: #30=↑X was an underline. Thus, Lindsey and I have added all of the third generation special characters ourselves, before sticking them out on the subdirectory <Original>. The only character already in these fonts that we changed is the single quote, #47. In first generation fonts, this was a symmetric single quote, but it is now a closing quote character; we redrew it so that it would look good opposite the opening quote character, which now appears at both code #7 and code #140.

Be warned that several other characters have changed substantially since first generation days: in particular, the code #55 (which you get by pushing the key to the right of zero) was a minus sign in first generation fonts, but is now a hyphen, so it has shrunk substantially. Meanwhile, the characters '+' at #53 and '=' at #75 have grown to a full em in width. Thus, the updated first generation Alto fonts don't correspond too closely to the printing fonts in some cases.

Why, you ask, didn't I go through and fix these characters as well, to make them correspond more closely to the current printing fonts? My excuse is rather involved, and has its origins in the differences between programs and English text. Since TimesRoman and Helvetica are intended as text fonts, it was a reasonable decision to change #55 from minus to hyphen, and to relegate minus to a control code: hyphens are more common in text. But when programming, minus signs are much more common; and in addition, most programming languages still think that #55 is a minus sign. This discrepancy is normally handled by using the font Gacha for programming purposes. Gacha hasn't changed since first generation days, and a #55 in Gacha is still a minus sign.

It transpires, however, that there are people in the world (in CSL, in fact) who don't like the way that Gacha looks, and prefer to stare at their programs in TimesRoman. For these people, it is better if the Alto character associated with #55 remain a minus sign, so that it will look balanced with plus and equal signs. Of course, if you print a program in TimesRoman, you will just have to live with hyphens instead of minus signs: nothing I can do about that. But I didn't want to give anyone the chance to say that my updates had made anything worse. Therefore, I left all of the old characters in the first generation Alto fonts alone, except for the closing quote. If you want the screen to look more like what comes off the printer, you should use the third generation Alto fonts discussed below. If you want to program in TimesRoman, then pull your fonts from <Original>.

Second generation Alto fonts: <AltoFonts>Thin>

When Ron Pellar produced Alto fonts for the second generation release, he chose to make his characters much thinner than those in the first generation fonts, and, as a result, these fonts are less legible. Although thinner, they are still not thin enough to be totally faithful to the printed page. Considering how thin they are, I am quite impressed by their legibility, and by how TimesRoman on the Alto screen really looks TimesRomanish.

Ron had already gone through these fonts and updated them to the third generation. He stores them fonts on [XEOS]<FontCenter>. The only changes that I meant to make to these fonts before putting them out on the subdirectory <Thin> was to fix obvious bugs that I happened to notice, such as accents out of place, and missing characters. I may have succumbed to temptation in a few

cases, however, and changed something which just looked a little wrong. If so, I have probably assumed the role of the apprentice "fixing" the work of the master (Pellar), and I apologize for any damage that I have wrought.

The thinnest of all: <AltoFonts>RoundedWidths>

How far can you go if you are willing to sacrifice style and legibility completely to achieve fidelity? Since a character in an Alto font is always an integral number of Alto pixels in width, the best that you could do is to make the width of each Alto character be the result of rounding its printing width to the nearest integral number of Alto dots. That was the principle behind a set of Alto fonts that Ron Pellar produced a while back, and stored on [XEOS]<ASD>Alto>. If you are using these fonts on your Alto screen, positions on your display and positions on the page will correspond to within the accumulated roundoff error. Since Ron had not bothered to update these fonts from the second to the third generation of special characters, Lindsey and I had a little more work to do on these, before we stuck them out on the subdirectory <RoundedWidths>.

One warning: to be true to the concept of rounded widths, the accent charcters in these Alto fonts are really zero dots wide. This causes problems for some systems, Bravo in particular. Hence, if you try and use these fonts with Bravo, the accents will come out as black rectangles even though the characters really are in the font. In all the other Alto fonts, the accents are one dot wide, to keep Bravo happy. Note that you can select a one dot wide character, although you have to look closely to do so. The truly zero width accent characters in the rounded widths Alto fonts don't bother SIL, by the way; I haven't checked out other systems.

Third generation Alto fonts: <AltoFonts> and <AltoFonts>EightBit>

As part of the third generation release, Pellar produced an entirely new set of Alto fonts. These are roughly as wide as the <Original> Alto fonts, but have a somewhat different feel. Not only did they have the third generation special characters when I retrieved them, they also have all of the other characters like hyphen, plus, and minus looking the way that they look in the printing fonts. I personally like the appearance of these fonts. Putting these factors together, I chose to put these fonts on the directory [Ivy]<AltoFonts> without any qualifying subdirectory; thus, after the cataclysm, these should be considered the "standard" Alto fonts.

Now for the details: third generation fonts have special characters not only in the ASCII control slots, but also associated with eight bit character codes, codes in the range #200 through #377. These eight bit characters are used by BravoX, and possibly by other systems, I don't know. But I do know that none of the standard CSL software for dealing with text or graphics can handle eight bit characters: it was all written with only seven bit characters in mind. Hence, it seemed pointless to put the eight bit characters into the new "standard" Alto fonts: for a few years, almost no programs will let you get at them anyway. Beyond that, the eight bit characters take up valuable space, and might very well tickle bugs in existing systems. Therefore, before putting these fonts out on [Ivy]<AltoFonts>, I removed all of the characters with eight bit codes. If you want a font that contains the eight bit characters, for BravoX or for some other reason, you can find them on the subdirectory <EightBit>.

There is also an issue about naming conventions for these fonts. I retrieved these fonts from [XEOS]<FontCenter>, the same directory from which I pulled the <Thin> fonts. Rather than use subdirectories as I have, Ron Pellar chose to distinguish these new Alto fonts by sticking an "E" in their name: for example, TimesRomanE10.al. I think that this was a lousy decision for several reasons. The characters in these fonts are wider than fidelity would demand, and the terms "expanded" and "extended" are used in typography for fonts with a wider face. One might assume that the "E" stands for either "Expanded" or "Extended". But if this is what the "E" is supposed to imply, then it should come after the point size number, not before it. After all, Condensed,

Regular, and Expanded are all legal face designators in the PARC font software, just like Bold and Italic. As supporting evidence for this point of view, let me note that whoever pulled TimesRomanE12.al from [XEOS] to put it on the directory [Maxc1]<Printing> chose to rename it to TimesRoman12E.al (and that person wasn't me!). On the other hand, it is perfectly conceivable that the "E" really designates "Extended" in the sense that the character set is extended to include the eight bit characters. If this was the intent, it is easier to see why the "E" would go before the number. But, in this case, an analogy with TimesRomanD would lead one to assume that TimesRomanE designated a different family than TimesRoman, which is false: the font on the printers is TimesRoman without the "E". In summary, it seems to me that the "E" in the font names is a poor idea; I chose to delete the "E" from the names of all of the third generation Alto fonts, and to distinguish them with the use of subdirectories as descibed above.

Circumflexes:

One types a circumflex accent in TimesRoman and Helvetica by typing both a grave accent and an acute accent. Since the accents in most Alto fonts have a spacing width of one rather than zero pixels, it makes a difference which of the grave and acute accents you type first. All of the updated Alto fonts on [Ivy] have been designed with the intent that the circumflex be typed in the order "↑K ↑E" rather than the reverse; this order allows the group of three accents to look better. Just remember that "K before E is the KEy", while "E before K is wrong: EeK!".

Alto Font Baselines:

Many of the oldest Alto fonts had their baselines off by one, according to the current conventions for Alto fonts. This off by one bug has been fixed in all of the updated fonts.

Different Faces:

The fonts produced by Pellar were mostly drawn with Bravo in mind, and Bravo synthesizes its own bold and italic from the vanilla Alto font. Other systems however, such as Draw, demand that you supply an Alto font in the face that you desire. There aren't very many Alto fonts in either bold or italic, and none in bold-italic. But you can find what there is, updated to the third generation of special characters, on [Ivy]<Altofonts>. The subdirectory <Original> has the most fonts in non-vanilla faces.

Different Families:

The text above was mostly written with the families TimesRoman and Helvetica in mind. Several versions of Alto fonts also exist for Gacha, Hippo, and Math in some sizes: they have also been updated and moved to [Ivy]<AltoFonts>. I put them on the subdirectory that seemed most appropriate. For other families, it is generally the case that at most one Alto font exists for each size; in these cases, I put that font on [Ivy]<AltoFonts> with the null subdirectory.

Inter-Office Memorandum

To	Whom it May Concern	Date	April 1, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	Ensuing font cataclysm	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>EnsuingCataclysm.bravo
[Maxc1]<Fonts>EnsuingCataclysm.press
[Ivy]<Fonts>Memos>EnsuingCataclysm.bravo
[Ivy]<Fonts>Memos>EnsuingCataclysm.press

The time has come for an update in CSL's font world. Sometime soon, I plan to replace many of the printing fonts in Clover's current font dictionary with new and better versions. The TimesRoman and Helvetica families will be replaced by Ron Pellar's latest release; and an entirely new set of TEX fonts produced by Don Knuth with Metafont will also make their debut. As the first step towards effecting this cataclysm, I am pleased to announce the release of a hoard of updated Alto fonts: every Alto font for TimesRoman and Helvetica that I could find over the InterNet has been updated to reflect the state of the world after the cataclysm. The details of this Alto font release are discussed below.

Like most pieces of progress, this cataclysm will cause some joy and some pain. I recommend that everyone read this message, to get a vague sense of what is going to happen. Experts may also wish to read one or more of a group of subsidiary memos that discuss various issues related to the cataclysm in greater detail. Here is a list of all of the documentation relating to the cataclysm:

EnsuingCataclysm (The text of this message.)
SpecialCharacters
TexFonts
AltoFontGuide
UniversityFonts

Each of these is available on [Maxc1]<Fonts> and on [Ivy]<Fonts>Memos> in both a ".Press" and a ".Bravo" version.

In addition, there are two other items of font documentation that you might want to look at right after the cataclysm. First, recall that the file [Maxc1]<Fonts>CloverFonts.Press always contains a tabular listing by family name, face, and size of all of the fonts currently installed on Clover, and on those printers who share Clover's font set (Lilac, Menlo, Kanji, Quake, and Rockhopper among others). But since the family name of a font doesn't help you much if you are looking for a particular character, I will be producing another file called [Maxc1]<Fonts>CloverCharacters.Press. For each family of fonts on Clover, this document will have a one page table that shows what printing character is associated with each character code. I hope that this file will be an adequate substitute for a true Clover font catalogue.

By the way, if you think that I have made some wrong decisions in planning this upcoming cataclysm, let me know; after all, I am still something of a novice as a font wizard.

Terminology:

For the purposes of this memo, a *font* is a means by which certain graphical shapes are associated with certain seven or eight bit numbers, called *character codes* (always expressed in octal, not decimal). Part of the design of any font is the specification of the correspondence between characters and their codes. Our Roman fonts, for the most part, are extensions of a particular coding scheme called *ASCII* (the American Standard Code for Information Interchange). Designed with teletypes in mind, ASCII sets aside about thirty character codes called the *control codes* for actions that a terminal might perform, such as tabbing to the next column, advancing to the next line, etc. Our Roman fonts have begun to use control codes for printing purposes. I will use the term *special characters* to refer to printing characters that have been associated with the ASCII control codes.

History:

It is helpful to think of the evolution of our fonts over the years in terms of *generations.* I refer to any font that was produced before October 1978 as a *first generation* font; their origin is shrouded in antiquity. The first generation fonts have only one special character: the ASCII control code #30=↑X is an underline (used by EARS). Ron Pellar's first big release of fonts in November of 1978 constituted the *second generation.* These fonts have about twenty special characters, mostly various ligatures, accents, spaces, and dashes. Ron finished the third generation of fonts in January of 1980. In these fonts, every seven bit code has either a printing role, or an active control function. In addition, printing roles have been given to about forty codes above #177, which I will call *eight bit codes.* The subsidiary memo SpecialCharacters lists the character code assignments of the various generations of fonts, and discusses some associated issues.

What will happen to Clover?

Almost all of the TimesRoman's and Helvetica's currently in Clover's data base are second generation. After the cataclysm, they will be third generation. This means that there will be a few more special characters associated with seven bit codes for you to use and enjoy. In addition, many little glitches will (finally!) go away: the rivers after lower-case 'x', the too large lower-case 'o', etc. In addition, there will be other new special characters associated with eight bit codes that probably won't do you any good, but won't do you any harm either.

Now for the bad news: there is one incompatible feature of this conversion from second to third generation. In second generation TimesRoman and Helvetica, the symbol minus sign is associated with two different control codes: both ↑N=#16 and ↑X=#30. (The character to the right of zero on the keyboard has code #55; in Timesroman and Helvetica, this code designates a hyphen, not a minus sign.) Third generation TimesRoman and Helvetica have stolen the code ↑N for a macron, the horizontal bar accent frequently used in dictionaries to indicate a long vowel. Hence, after the cataclysm, you must use the code ↑X if you want a minus sign.

In addition, Press files prepared before the cataclysm using ↑N's for minuses will not print properly after the cataclysm. This will have to be dealt with on a file by file basis. One way, of course, is to reproduce the Press file from more primitive sources, after editing those sources to use the ↑X code rather than ↑N to specify a minus sign. There is an alternative, however: a program could be written to scan a Press file, and replace all of the ↑N's in text strings with ↑X's. Such a program would automatically massage pre-cataclysm Press files so that they would print correctly after the cataclysm. I will write this program if I perceive substantial demand: let me know.

The Math font is rather a different story. When Ron Pellar produced the second generation fonts, he choose to change many of the character code assignments of the Math font. This blatant incompatibility with the past generated so much protest that CSL decided to keep Clover running

with first generation Math. In fact, Clover is still running with first generation Math. In addition, Ron was prevailed upon to rescind these character code changes when the third generation Math came out. Third generation Math is exactly like first generation Math, except for the addition of some valuable new characters in control slots. Second generation Math (described, unfortunately, in the Novermber 1978 edition of the Alto User's Handbook) should be forgotten as soon as possible. The cataclysm will convert Clover from first to third generation Math; this conversion is believed to be 100% backward compatible. For details of the new characters and the character mapping, see the SpecialCharacters memo.

The only other fonts substantially affected by the cataclysm will be the TEX fonts. Don Knuth is just now using Metafont to complete the design of the final fonts for new editions and volumes of *The Art of Computer Programming.* These new fonts are different enough from the old ones that they will be treated by the cataclysm as entirely new fonts, rather than simply as new versions. The old TEX fonts will stay around for a reasonable period in the interests of backward compatibility, after which they will go away. We are also taking advantage of the cataclysm to change the way that we handle the sizes of TEX fonts; for details, read the TexFonts memo.

Fonts for Altos:

The biggest issue throughout the history of Alto font construction has been that of character width. Since the Alto display is a low-resolution bitmap, it is difficult to draw small characters that look good; furthermore, it is especially difficult to draw characters that are as thin as they are supposed to be and look good. In most Alto fonts, therefore, the characters are actually much thicker than their nominal size; that is why they all crowd together in Hardcopy mode. But some Alto fonts are thinner than others.

First generation Alto fonts are those found on the directory [Maxc1]<AltoFonts>; being first generation, they have only one special character, an underline in the ↑X slot. They are also fairly wide. When Ron Pellar produced the second generation of printing fonts, he also produced a set of Alto fonts that are substantially narrower. This makes them more faithful to the printed page, but harder to read. Public opinion in CSL found these second generation Alto fonts unacceptable. Therefore, CSL has continued to use the first generation Alto fonts, even though these fonts were not updated to include the second generation special characters.

For the third generation release, Ron recently produced an entirely new set of Alto fonts that are roughly as wide as the first generation Alto fonts. In my personal opinion, these third generation Alto fonts are fairly good looking; furthermore, they come in lots of sizes and they have all the right special characters. On the other hand, it is well known that there is no accounting for taste, and that change is intrinsically evil. Hence, I anticipated substantial pressure from CSL'ers not to be torn away from the first generation Alto fonts that they have used for so long.

I decided that the only way to keep everyone happy was to allow everyone to roll their own. In particular, I took all three generations of Alto fonts as well as special set of very thin Alto fonts that Ron produced for ASD a while back, and brought all of them up to date by adding the third generation special characters. These new Alto fonts are all available now on the directory [Ivy]<AltoFonts>. The different versions of each font are distinguished by the use of subdirectories. For the names of these subdirectories and what they mean, read the AltoFontGuide memo. To date, I haven't done anything about the Alto fonts on Maxc1. I propose that, as part of the cataclysm, the updated first generation Alto fonts be placed on [Maxc1]<AltoFonts> while Pellar's third generation Alto fonts be made available on [Maxc1]<Printing>.

By the way, there is one minor detail about these fonts that deserves comment here, concerning the circumflex accent. To save a character code, the recent TimesRoman and Helvetica fonts have been designed so that overprinting a character with both the acute accent (found at #13=↑K) and the

grave accent (found at #5=↑E) will result in a reasonable circumflex accent. If the spacing width of the accent characters were exactly zero, then it wouldn't matter in which order you put the ↑E and ↑K. But, to keep Bravo happy, the spacing widths of the accents must strictly positive. In the printing fonts, the accent widths, although positive, are small enough that the order of the ↑E and ↑K is almost immaterial. In the Alto fonts, however, the accents are one whole Alto pixel wide; as a result, the order of the ↑E and ↑K accents does matter to the Alto font designer. Furthermore, it is not just an arbitrary convention! Since the circumflex goes up first and then down, an Alto font will look better if it was designed to have "↑K ↑E" come out as circumflex than if it was designed for a "↑E ↑K" circumflex. Therefore, all of the updated Alto fonts on [Ivy] have been built with the assumption that the acute accent will be typed before the grave accent for a circumflex: just remember that "K before E is the KEy", while "E before K is wrong: EeK!".

University Impact:

This cataclysm will have less of an impact on the universities in the grant program, since they have been operating with third generation fonts from the very beginning. I will take advantage of the cataclysm, however, to build a new printing font dictionary for the universities that contains the new sizes and faces that Pellar has produced. In addition, this new dictionary will contain several random fonts like Sail and Apl that were omitted in the first release. Once the updated Alto fonts have been moved to Maxc1, the universities will be encouraged to switch over to them. For more details, see the last of the auxiliary memos: UniversityFonts.

Inter-Office Memorandum

To	Whom it May Concern	Date	April 17, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	Kerned Strike Fonts (Revised version)	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>KenrnedStrikes.bravo
[Maxc1]<Fonts>KenrnedStrikes.press

It's time for a new font format! The Strike format for font files was devised to permit the graceful use of BITBLT for writing characters onto the Alto screen; the authoritative description of this original Strike format can be found in the memo *Font Representations and Formats* by Bob Sproull, filed on

[Maxc1]<PrintingDocs>FontFormats.Press.

The original Strike format, however, has a serious difficulty: it does not distinguish between the spacing width of a character, and the width of that character's bounding box. Let us say that a character *kerns to the left* if its raster includes bits in columns to the left of the character's origin. A character is said to *kern to the right* if its raster includes bits in columns to the right of the end of its width vector (the origin of the next character). The original Strike format is incapable of handling characters that kern in either direction. The newer PARC fonts contain lots of zero-width accent characters that are intended to overprint the following character. These accent characters overhang the ends of their width vectors by quite a bit, that is, they kern heavily to the right; hence, conventional Strike format can't handle them.

There are basically two ways to adjust Strike format so that it can handle characters that kern. Consider left kerning, for example. One possibility might be called the *padding approach.* The idea here is to pad every character raster in the strike body with enough blank columns so that all of the overhanging bits fit into those padding columns.

A padding approach to the left kerning problem would work like this. We would look through the entire font, and determine the size of the largest left kern of any character; say that it is four columns. Thus, at least one character's bounding box includes four columns to the left of the origin, while no character overhangs to the left of the origin by more than four columns. When putting character rasters into the Strike array, then, we include four columns for every character to the left of that character's origin. We also find a field somewhere early in the file to put the number '4'. A user of the resulting file would paint a character on the display by BitBlt'ing the entire character raster, padding columns and all. The first column of the destination of the BitBlt would be given by the formula (<desired origin>−4).

A symmetric padding strategy could be used to handle right kerning. But any padding strategy has problems. The padding columns appear in every character's raster; they take up space in the file, and in memory, and they slow down the process of painting the character, since they are included in the BitBlt's.

The other approach to the kerning problem might be called the *auxiliary table approach.* In this scheme, we store the character rasters into the strike body by their bounding box dimensions, and then we use an auxiliary table of small integers to tell the location of the origin relative to the

bounding box, and the length of the width vector. The auxiliary table approach seems to be a better choice than a padding approach. The purpose of this memo is propose a new Strike file format that uses the auxiliary table approach to handle both left and right kerning. If you object to the following plan, or have suggestions to improve it, please get in touch with me as soon as possible.

History:

The definition of Strike file format in Sproull's memo *Font Representations and Formats* suggests the use of a padding scheme to handle left kerning. As I read it, this memo states that a negative integer in the xoffset word implies that every character raster in the strike is padded by (−xoffset) columns on the left. To the best of my knowledge, no one ever wrote any programs that would correctly produce or consume such a left-padded strike font. In addition, left kerning doesn't happen to be nearly as common in PARC fonts as right kerning anyway. We could, of course, legislate against left kerning completely, and start devoting the xoffset word to a right kerning scheme. In fact, this is what the current version of PrePress does when the Kerned flag is set in MakeStrike.

But thinking things over, it seems to be a better plan to take advantage of the fact that no padding scheme has been widely used, and drop all such schemes in favor of a format based on the auxiliary table approach. From now on, if you want to handle kerning, you should use the new KernedStrike format, and its auxiliary tables.

Some terminology:

The following section is a substitute for section 7.2 of the *Font Representations and Formats* memo. Before we start on that, though, let me review a little terminology from that memo, in case the reader is rusty. The origin of a charcter is a reference point located at the corner where four pixels touch. The width vector of a character is a two-dimensional vector that specifies the desired displacement from the origin of the current character to the origin of the next character in a string. The components of the width vector are written Wx and Wy. Wx is always nonnegative; all of the font formats intended for the Alto screen assume in addition that Wx is an integral number of pixels, and that all characters have a Wy of zero. The bounding box of the black pixels in the character raster is a rectangle with width BBdx and height BBdy. These numbers are always nonnegative. They are both zero if the character has no black bits in its raster (such characters are called *empty characers*), and are both positive in all other cases. If we use the origin to define a Cartesian coordinate system on the plane, the lower left corner of the bounding box is located at the point <BBox, BBoy>. Thus, the left edge of the bounding box is located BBox units to the right of the origin (left if BBox is negative), while the bottom of the bounding box is located BBoy units above the origin (below if BBoy is negative). Empty charcters have BBox and BBoy equal to zero, by convention. Finally, if all of the characters in the font are superimposed with their origins coincident, the dimensions of the resulting bounding box are called FBBdx, FBBdy, FBBox, and FBBoy respectively.

7.2 (New version) STRIKE format:

There are four kinds of files in the Strike class: a PlainStrike file (conventional extension .Strike), a KernedStrike file (conventional extension .KS), a PlainStrikeIndex file (conventional extension .StrikeX), and a KernedStrikeIndex file (conventional extension .KSX). In a PlainStrike file, the individual rasters of the characters are assembled in ascending order of character code into one large raster, called the *strike*. The baselines of the characters are aligned, and the origin of each character is made coincident with the end of the width vector of the preceding character. The PlainStrike file also contains a table indexed by character code that points to the leftmost column of the raster for each character in the strike. Warning: since the rasters in a PlainStrike file are positioned by their

origins and width vectors, it must be the case that all of the black bits of the character lie between these two bounds. No character may include bits to the left of its origin (left-kerning) or the right of the end of its width vector (right-kerning).

A KernedStrike file handles kerned characters, and does so in the following way: the individual rasters are put into the strike by their bounding box widths. Just think about taking the bounding boxes of all of the characters, lining up their baselines, and packing them tightly into one long raster array; in this format, there are no blank columns between characters in the strike. A KernedStrike file has three additional tables, indexed by character code. One gives the position in the strike of the first column of the character's raster, which is also the leftmost column of its bounding box. The other two tables consist of small integers that specify the left-to-right location of the origin with respect to the bounding box, and the length of the width vector.

A StrikeIndex is essentially a table that maps character codes into <strike, code> pairs, together with the associated strikes. An index can be used to achieve sharing if several character codes map to the same <strike, code> pair, and hence refer to the same raster. Or it can help to save space, by grouping the rasters into several strikes to save top and bottom scanlines. [By the way, to the best of my knowledge, no one has ever used StrikeIndex format.]

PlainStrike and KernedStrike files have the following format:

```
structure PlainStrike:
  [
  @StrikeHeader          // header common to all Strike files
  @StrikeBody            // the actual strike
  ]

structure KernedStrike:
  [
  @StrikeHeader          // header common to all Strike files
  @BoundingBoxBlock      // dimensions of the font bounding box
  @StrikeBody            // the actual strike
  @WidthBody             // table of width data
  ]

structure StrikeHeader:
  [
  format word =
              [
              oneBit bit     // always =1, meaning "new style"
              index bit      // =1 means StrikeIndex, =0 otherwise
              fixed bit      // =1 if all characters have same value of Wx, else =0
              kerned bit     // =1 if KernedStrike, =0 if PlainStrike
              blank bit 12
              ]
  min word               // mimimum character code
  max word               // maximum character code
  maxwidth word          // maximum spacing width of any character = max{Wx}
  ]

structure BoundingBoxBlock:
  [
  FBBox                  // as defined above
  FBBoy                  // as defined above
```

```
  FBBdx                    // as defined above
  FBBdy                    // as defined above
  ]

structure StrikeBody:
  [
  length word              // total number of words in the StrikeBody
  ascent word              // number of scan-lines above the baseline, which is
                           //   normally max{BBdy+BBoy} over the chars in this strike
  descent word             // number of scan-lines below the baseline, which is
                           //   normally max{(-BBoy)} over the chars in this strike
  xoffset word             // always =0 [used to be used for padding schemes]
  raster word              // number of words per scan-line in the strike
  bitmap word raster*height // the bit map, where height=ascent+descent=FBBdy
  xinsegment ↑ min, max+2 word // pointers into the strike, indexed by code
  ]

structure WidthBody:
  [
  widthtabl ↑ min, max+1 @WidthEntry   // spacing information, indexed by code
  ]

structure WidthEntry:
  [
  spacing word =           // the entire spacing word will be =(-1) (both bytes =377b)
                           // to flag a non-existent character, else the bytes are:
            [
            offset byte    // =BBox-FBBox
            width byte     // =Wx
            ]
  ]
```

The "bitmap" entry is one large bit map; there are height=ascent+descent scanlines in the bitmap, each of which is raster words long. Unless something funny is going on, ascent will be simply FBBdy+FBBoy, while descent will be simply (-FBBoy).

The font includes characters for some of the ASCII codes from min through max inclusive. The bitmap includes a dummy character associated with the charcter code (max+1), which can be displayed for any non-existent character.

A PlainStrike works as follows: Given a character code c, in the range [min, max], we first compute:

```
xLeft ← xinsegment ↑ c;
xRight ← xinsegment ↑ (c+1);.
```

If xLeft=xRight, then c is a non-existent character in the current font, and should be replaced by the raster with code (max+1). Otherwise, the columns of the bitmap from xLeft through (xRight-1) inclusive contain the raster for character c, and the width of charcter c is Wx=(xRight-xLeft).

A KernedStrike works a little differently. We first compute xLeft and xRight as above, and also compute

```
Spacing ← WidthTable ↑ c;.
```

If Spacing=(-1), then c is a non-existent character in the current font, and should be replaced by

the dummy character at (max+1). Otherwise, the columns of the bitmap from xLeft through (xRight−1) constitute the bounding box of the raster of character c. In this case, we decompose the Spacing value into its two bytes:

Offset ← Spacing<<WidthEntry.offset;
Width ← Spacing<<WidthEntry.width;.

Now assume that we want to paint the character c starting at destination column xDest. The source of the BitBlt is columns xLeft through (xRight−1) of the bitmap inclusive. We can compute the proper destination from xDest, Offset (which =BBox−FBBox), and the FBBox word of the BoundingBoxBlock: the first column of the destination is (xDest+Offset+FBBox)=(xDest+BBox). Note: the offset portion of the WidthEntry was chosen to be (BBox−FBBox) rather than BBox itself, since the former quantity is always nonnegative, while the latter quantity can have either sign; and signed 8-bit numbers are a pain in the ass. Finally, we replace xDest by (xDest+Width) to prepare for the painting of the following character.

Two fine points concerning KernedStrikes: A non-existent character is flagged by a (−1) value in the WidthTable. Since a non-existent character doesn't have a bounding box, the xLeft and xRight entries for such a character will be equal. But there can also be perfectly legal characters for which xLeft=xRight; in particular, all of the empty characters will have this property: figure space, em quad, word space, etc. If you are painting an empty character, there is no need to actually perform the BitBlt, since the rectangle being Blt'ed would have zero width. All that must be done is to replace xDest by (xDest+Width) to make the space happen. Secondly, some extra efficiency can be gained when using a KernedStrike font by keeping track of the quantity (xDest+FBBox) in the character-painting loop, instead of xDest itself. This moves one addition out of the inner loop.

Finally, it is time to say a few words about StrikeIndex format: a StrikeIndex is simply an index at the front of some StrikeBodies.

```
structure PlainStrikeIndex:
  [
  @StrikeHeader                    // common header
  maxascent word                   // maximum ascent of all the strikes
                                        // [probably =FBBdy+FBBoy]
  maxdescent word                  // maximum descent of all the strikes
                                        // [probably =(−FBBoy)]
  nStrikeBodies word               // the number of strike bodies
  map ↑ min,max+1 @mapEntry        // table of <strike, code> pairs, dummy at max+1
  bodies ↑ 1, nStrikeBodies @StrikeBody // the strike bodies themselves
  ]

structure KernedStrikeIndex:
  [
  @StrikeHeader                    // common header
  @BoundingBoxBlock                // bounding box data for the entire font
  maxascent word                   // maximum ascent of all the strikes
                                        // [probably =FBBdy+FBBoy]
  maxdescent word                  // maximum descent of all the strikes
                                        // [probably =(−FBBoy)]
  nStrikeBodies word               // the number of strike bodies
  map ↑ min,max+1 @mapEntry        // table of <strike, code> pairs, dummy at max+1
  bodies ↑ 1, nStrikeBodies @StrikeBody // the strike bodies themselves
  @WidthBody                       // table of width data
  ]

structure mapEntry:
```

```
[
missing bit 1          // =1 if character is non-existent, else =0
strike bit 7           //which strike in range [0:127]
code byte              // which code
]
```

In a StrikeIndex font, all of the StrikeBodies have implicit min values of zero; the max value is unimportant, as the map will never generate a reference outside the range. The individual StrikeBodies do not have separate pictures for illegal characters; instead, the (max+1) entry in the global map defines a single dummy picture. Non-existent characters in the range [min, max] are indicated in the global map by a mapEntry that specifies a strike number larger than 127=177b, that is, by the sign bit of the map entry being 1. In KernedStrikeIndex fonts, non-existent characters will also be indicated by having a WidthEntry of (−1).

In StrikeIndex fonts, the ascent and descent words in each StrikeBody give the dimensions of that particular StrikeBody; thus, they probably are the y dimensions of the bounding box of those characters that are included in that StrikeBody, rather than of the entire font.

BitBlt modes:

There are evidently lots of programs in the world that paint charcters on the screen by calling BitBlt in Replace mode, in which the new bits simply smash whatever used to be at the destination. If you want to handle characters that kern, you simply can't do this! The bounding boxes of successive characers may actually overlap, and hence a Replace Blt might overwrite valuable bits. If you want kerning specified in a KernedStrike font to work, you must use one of the other BitBlt modes: Paint, Erase, or Invert.

Conversion issues:

I propose the following plan. In the near future, some combination of Doug Wyatt and myself will modify PrePress 1.12 to produce a PrePress 1.13. This new PrePress will include a MakeKS command, that converts a font from .AC format to .KS format, and a ReadKS command, that converts in the opposite direction. The MakeStrike command will remain, to convert from .AC format to the original Strike format. However, the Kerned flag will be replaced by a new flag called 'Clipped'. And the code that implements the MakeStrike command will be altered to treat overhanging characters as follows. If any character of the .AC font kerns to the left, the MakeStrike command will fail, as it does currently. Characters will be allowed to kern to the right. If the Clipped flag is true, the portion of the bounding box that overhangs the end of the width vector will be ignored. If the Clipped flag is false, the width of any right-kerned characters will be artificially increased just enough to eliminate the overhang. Thus, calling MakeStrike on a font with right kerning will have one of two effects: if Clipped is true, the resulting Strike font will space correctly but the overhanging portions of characters won't print. If Clipped is false, the overhanging portions will print, but the characters will be spaced too far apart. Neither of these two possibilities is very pleasant, of course; but if you want kerned characters to work, you should use .KS format.

Note that the xoffset word of all StrikeBodies produced under the new rules will always be zero, no matter what file format is involved. Non-zero values of xoffset were originally reserved for handling kerning by padding schemes; since padding is now handled by the auxiliary table approach, the xoffset word is not needed.

PrePress 1.13 will also include new code to handle the exotic face byte values that are being introduced as part of the upcoming font cataclysm.

Once PrePress 1.13 is up and running, we will take all of the .AL fonts on [Ivy]<AltoFonts> and produce a .KS version for each of them, sticking all of those fonts on [Ivy]<AltoFonts> as well. That should give us plenty of KernedStrike fonts to play with, as people start to produce software that uses the new format.

Inter-Office Memorandum

To	Whom it May Concern	Date	July 19, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	Special characters over the years	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>SpecialCharacters.bravo
[Maxc1]<Fonts>SpecialCharacters.press
[Ivy]<Fonts>Memos>SpecialCharacters.bravo
[Ivy]<Fonts>Memos>SpecialCharacters.press

This memo discusses the allocation of character codes over the years in the fonts TimesRoman, Helvetica, and Math, and also deals with some related issues persuant to the July 1980 font cataclysm. In this memo, I will merely list the assignments of the special character codes Since I want the reader to be able to print this memo before the cataclysm, I was not able to include any demonstration of what the new special characters of the third generation actually look like.

Since the cataclysm has already occurred, there is a new document on [Ivy]<Fonts> called CloverCharacters.Press. This is a sort of companion to the documentation file CloverFonts.Press on the same directory. Where CloverFonts simply lists the names, sizes, and faces of the fonts on Clover, CloverCharacters has a one page table for each family that shows what printing character is associated with each character code. CloverCharacters only discusses the post-cataclysm truth, and hence doesn't give the sense of change and history provided by this memo. On the other hand, CloverCharacters has pictures instead of words, and CloverCharacters includes all of the families on Clover, not just TimesRoman, Helvetica, and Math.

TimesRoman and Helvetica:

Let's ignore Math for the present, and concentrate on TimesRoman and Helvetica. Almost every Roman font agrees on the meanings of almost all of the character codes from #40 through #176. These character codes are defined in PARC Roman fonts by the following rough equivalent of ASCII:

#40 = space <see discussion below>
#41 = exclamation point
#42 = symmetric double quote
#43 = number sign, sharp sign, pound sign
#44 = dollar sign
#45 = per cent sign
#46 = ampersand
#47 = single quote <see discussion below>
#50 = left parenthesis
#51 = right parenthesis
#52 = asterisk
#53 = plus sign
#54 = comma
#55 = minus sign or hyphen <see discussion below>
#56 = period
#57 = slash

#60 through #71=digits 0 through 9
#72=colon
#73=semicolon
#74=left angle bracket (*not* less than sign)
#75=equal sign
#76=right angle bracket (*not* greater than sign)
#77=question mark
#100=at sign
#101 through #132=upper case A through Z
#133=left square bracket
#134=backslash
#135=right square bracket
#136=up arrow
#137=left arrow
#140=<see discussion below>
#141 through #172=lower case a through z
#173=left curly bracket
#174=vertical bar
#175=right curly bracket
#176=tilde symbol (not accent).

The history of character sets at PARC has mostly involved the fate of the other ASCII character codes.

First Generation:

In the first generation fonts:
#40 was the only space there was
#47 was a symmetric single quote
#55 was a minus sign
#140 was not defined.

In addition, there was one special character: #30=↑X was defined to be an underline accent character. Gacha is an example of a first generation font that has survived to the current day.

Second Generation Changes:

The second generation added nineteen special characters, and affected a few of the basic codes as well. The changes to the basic characters were:
#40 became a space designed to go between words
#47 became a closing single quote
#55 became a hyphen
#140 was still undefined.

The special characters introduced were:
#2=↑B=upsidedown question mark
#3=↑C=a cedilla accent on a lower case c
#4=↑D=umlaut accent
#5=↑E=grave accent
#6=↑F=ff ligature
#7=↑G=opening single quote
#10=↑H=upsidedown exclamation point
#13=↑K=acute accent
#16=↑N=minus sign
#17=↑O=em quad
#20=↑P=tilde accent
#21=↑Q=ffi ligature

#22=↑R=ffl ligature
#23=↑S=em dash
#24=↑T=fi ligature
#25=↑U=fl ligature
#26=↑V=en dash
#30=↑X=minus sign, duplicating #16
#31=↑Y=figure space, a space just as wide as a digit
#34=↑\=en quad.

Third generation:

In the third generation release, the region of character code space from #1 through #37 was pretty much filled up, and extra characters began to find there way into the eight bit character codes, those between #200 and #377. The changes to the basic codes were:

#40 remained a word space
#47 remained a closing single quote
#55 remained a hyphen
#140 now duplicates #7, an opening single quote.

This last change was an excellent choice from a public relations point of view, since much of Arpa community was already committed to this convention.

In third generation fonts, the first 40b character codes are allocated as follows:

#0=ASCII null
#1=↑Λ=hacek accent, upsidedown circumflex
#2 through #10 are as in second generation
#11=↑I=ASCII tab, used by Bravo
#12=↑J=ASCII line feed, treated specially by Bravo
#13=↑K=acute accent, as in second generation
#14=↑L=ASCII form feed, page break in Bravo
#15=↑M=ASCII carriage return, used by Bravo
#16=↑N=macron accent (!Unlike second generation!)
#17 through #26 are as in second generation
#27=↑W=breve accent
#30=↑X=minus sign (no longer duplicated at #16)
#31=↑Y=figure space, as in second generation
#32=↑Z=ASCII end-of-file and Bravo paragraph break
#33=↑[=ASCII Escape, used by Bravo
#34=↑\=en quad, as in second generation
#35=↑]=low dot accent
#36=↑↑=duplicate of tilde accent at #20, since #20 has been stolen by some Bravo's for something
#37=↑←=circle accent.

The change of #16=↑N from one of the minus signs to a macron accent is the only incompatible feature of the change from the second to the third generation.

As the previous listing indicates, the seven bit codes are now pretty well filled up; from #40 through #176 are the basic codes discussed at the beginning, and #177 is ASCII delete. Large-scale expansion of charcter sets must therefore either involve introducing new fonts or giving printing meanings to eight bit character codes. Here is the list of the eight bit codes that Pellar choose to define in the third generation fonts:

#200=umlaut on an upper case A
#201=umlaut on an upper case O
#202=circle accent on an upper case A
#220=ff ligature

#221=ffi ligature
#222=ffl ligature
#223=fi ligature
#224=fl ligature
#230=en quad
#231=em quad
#232=figure space
#233=en dash
#234=em dash
#235=minus sign
#236=en leader
#237=5-em space
#240=umlaut on a lower case a
#241=umlaut on a lower case o
#242=cirlce accent on a lower case a
#243=umlaut on a lower case u
#244=cedilla accent on a lower case c
#250=umlaut accent
#251=circle accent
#252=grave accent
#253=acute accent
#254=circumflex accent
#255=tilde accent
#256=breve accent
#257=macron accent
#260=cent sign
#261=Sterling sign
#265=star
#266=section sign
#267=bullet
#270=dagger
#271=double dagger
#272=paragraph sign
#274=plus-minus sign
#275=upsidedown question mark
#276=upsidedown exclamation point
#277=underscore
#337=non-required hyphen
#350=hacek accent
#351=low dot accent

By the way, the accents associate with the #250 through #257 and #350 and #351 codes are not zero width; they should only be used by programs that are smart enough to center them over the accented character.

Some thoughts about the eight bit characters:

From the CSL point of view, it is rather unfortunate that interesting characters are now beginning to be associated with eight bit character codes. Much of CSL's software world was written with only seven bit character codes in mind, and probably won't be changed in the near future. Consider Tex for example: this document compiler is used by a community of people at PARC. But Tex was written on a PDP-10, and has the assumption that character codes are seven bits long burned into its guts. This means that whatever pretty new characters Pellar sticks up in the eight bit region will probably never be accessible from Tex.

Above and beyond this long term worry, these eight bit characters have affected the plan for the font cataclysm. As described in the AltoFontGuide memo, I chose to strip the eight bit characters out of the standard Alto fonts. In addition, there are two different Fonts.Widths files for use after the cataclysm: the standard one ([Ivy]<Fonts>Fonts.Widths) has the eight bit characters stripped out, while the other ([Ivy]<Fonts>EightBit>Fonts.Widths) has them left in. Producing this stripped-down width dictionary demanded a little hacking on my part, but consider the alternative. The only other choice would have been to go over every program in the CSL software world that reads Fonts.Widths, and check that they can correctly discard width information about characters with eight bit character codes. And just consider the list of affected programs: Bravo 7.5, SIL, PressEdit, Laurel—even Pub! The complexity of having two different width dictionaries around pales into insignificance before the unpleasantness of hacking on Pub.

On the other hand, I plan to leave all of the eight bit characters in the dictionary of printing fonts. The printing servers and such font systems as Prepress have already been checked out on eight bit characters, and handle them correctly..

Math:

We bring this memo to a close by recounting the sad history of the Math font. The first generation of the Math font had the character map:

#40=printing symbol for a space
#41=dagger
#42=degree symbol
#43=infinity symbol, lazy eight
#44=cent sign
#45=division sign, elementary school version
#46=logical and, an A without the bar, lattice meet
#47=equal sign with dot above
#50=<unassigned>
#51=<unassigned>
#52=high dot
#53=plus-or-minus sign
#54=contains as an element, such that
#55=minus-or-plus sign
#56=three dots meaning therefore
#57=slash in a circle
#60=circle
#61=box
#62=triangle with flat side on the bottom
#63=diamond
#64=plus in a circle
#65=minus in a circle
#66=multiplcation x in a circle
#67=angle sign
#70=star
#71=high dot, darker than #52
#72=section sign
#73=black slug
#74=less than or equal sign
#75=not equal sign
#76=greater than or equal sign
#77=upsidedown question mark
#100=in care of
#101=for all, upsidedown A

#102=is an element of
#103=blackboard boldface C, the complex numbers
#104=nabla, upsidedown upper case Greek delta
#105=there exists, backwards E
#106=double dagger
#107=is a proper subset of
#110=is not a proper subset of
#111=is a subset of
#112=contains as a proper subset
#113=does not contain as a proper subset
#114=contains as a subset
#115=printing symbol for carriage return
#116=is not an element of
#117=the null set
#120=is proportional to
#121=right and left arrows with right on top, reversible reaction
#122=blackboard boldface R, the real numbers
#123=wavy equals
#124=is perpendicular to
#125=union
#126=logical or, a V shape, lattice join
#127=three bar equal
#130=a multiplication x
#131=arrow starting out rightwards, then pointing down
#132=solid triangle pointing rightward
#133=curly left angle bracket
#134=thick slash
#135=curly right angle bracket
#136=down arrow
#137=right arrow
#140=<unused>
#141=aleph
#142=wavy right arrow
#143=copyright symbol, small C in a circle
#144=partial sign
#145=equal sign with top bar wavy
#146=double-headed arrow
#147=double shafted right arrow
#150=slashed h, for Planck's constant
#151=left ceiling bracket
#152=right ceiling bracket
#153=left floor bracket
#154=right floor bracket
#155=double vertical bar
#156=logical negation sign
#157=small circle operator, composition
#160=T on its side, vertical bar on the left
#161=double shafted T on its side, vertical bar on the left
#162=registered trademark symbol, small R in a circle
#163=northeast arrow
#164=northwest arrow
#165=southwest arrow
#166=southeast arrow
#167=much less than sign

#170=much greater than sign
#171=intersection
#172=T on its side, vertical bar on the right
#173=the fraction 1/4
#174=the fraction 1/2
#175=the fraction 3/4
#176=right and left arrows with one-sided heads, left on top.

When Ron Pellar prepared the second generation of Math, he added a few new characters:
#2=↑B=less than sign
#3=↑C=greater than sign
#4=↑D=a one point space, for use in equations
#50=a prime, thin enough to be doubled up.
#51=upper case Greek pi, to reduce the need for Hippo.
In addition to making these additions, Ron decided to try replacing thirteen of the original characters with more useful printing roles. This incompatibility generated such storms of protest that Ron decided to rescind these replacements in the third generation of Math. If you really want to know what these characters were, you should look at the November 1978 or September 1979 edition of the Alto User's Handbook, the Bravo font page. Unfortunately, that page was prepared using second generation Math. I am not going to mention those replacement characters here, because it is my fond hope that all memory of them should vanish from off the face of the earth.

On to third generation Math! The following list describes the character code assignments of third generation Math where they differ from the first generation:
#1=↑A=upper case Greek pi
#2=↑B=less than
#3=↑C=greater than
#4=↑D=a one point space
#5=↑E=symbol for pound Sterling
#6=↑F=integral sign
#7=↑G=contour integral sign
#10 through #12 unassigned
#13=↑K=paragraph symbol
#14 through #16 unassigned
#17=↑O=bullet
#20 through #22 unassigned
#23=↑S=upper case Greek sigma
#24 and #25 unassigned
#26=↑V=three dots meaning because
#27 through #37 unassigned
#40 through #47 as in first generation
#50=prime as in second generation
#51=radical symbol, square root symbol
#52 through #137 as in first generation
#140 unassigned
#141 through #176 as in first generation.

Concluding remarks:

All existing Alto fonts for TimesRoman, Helvetica, and Math have been brought up to date with the third generation special characters, and put on the directory [Ivy]<AltoFonts>; see the AltoFontGuide memo. Clover was running with second generation TimesRoman and Helvetica and first generation Math; the cataclysm has brought all three families up to the third generation.

Inter-Office Memorandum

To	Whom it May Concern	Date	July 19, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	Cataclysm impact on TEX fonts	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>TexFonts.bravo
[Maxc1]<Fonts>TexFonts.press
[Ivy]<Fonts>Memos>TexFonts.bravo
[Ivy]<Fonts>Memos>TexFonts.press

Don Knuth just finished up the Metafont design of the family of fonts that will be used in the near future to typeset the second edition of Volume II of *The Art of Computer Programming.* I originally hoped to make those fonts available to PARC users of TEX as part of the font cataclysm announced on April 1, 1980. Unfortunately, I am not going to meet that deadline: the non-TEX portions of that font cataclysm happened on July 18, 1980, but the TEX portions won't happen for a little while yet. This documnt discusses the plans for the TEX font cataclysm that will occur later on. As part of this later update, we are planning to improve the manner in which font size is handled for Tex fonts.

Physical Size versus Logical Size:

What is the difference between a 9 point and a 12 point font in the same family? In the world of PARC font software, these two fonts are very closely related: in particular, you can get the 12 point font by merely scaling up the 9 point font by 1/3 in each dimension. Or you can get the 9 point by scaling the 12 point down by 25%. To put it another way, different sizes of the same family at PARC are strictly proportionally related. This proportionality has several advantages: first, the rasters for the characters can be produced by scaling their spline definitions, and then scan converting. In addition, a single table of width information can be scaled appropriately to give the widths of any size font in the family.

On the other hand, strictly proportional scaling is only an approximation to what a typographer really desires. It turns out that, as the height of a font decreases, the widths of characters should decrease less than proportionally. The thickness of the character strokes and the sizes of the serifs should also change non-proportionally as the height changes. The designers of the PARC software world realized this, of course, but chose to stick with strictly proportional scaling anyway, because of the programming benefits it provides. While proportional scaling is not perfect, it provides good results as long as the splines are not scaled up or down too far. When fonts of one family are needed that cover a wide range of sizes, the range can be broken into subranges, and a different set of splines produced for each subrange. In fact, this has already happened: TimesRomanD and HelveticaD are font families whose splines were drawn with the larger sizes in mind.

In Metafont, however, the width, stroke thickness, and serif sizes of each font are set independently to achieve the best typographic result. There is no assumption of proportionality. As an immediate consequence, the widths of each different size of each Tex font must be stored separately: this explains why Fonts.Widths files for Tex fonts are so large.

The non-proportionality of the Tex fonts wouldn't be as much of a hassle if it weren't combined

with another factor: people's desire to print magnified documents. Suppose that you are preparing a proceedings article that should appear in 9 point type. Also suppose that the printers of this proceedings will photographically reduces whatever masters you give them, say by 25%. If you are working with PARC fonts, you can allow for this reduction by choosing to print your document in 12 point type. Reducing 12 point TimesRoman by 25% will produce 9 point TimesRoman. But with Metafont, this is no longer the case. If you want the final result to be CMR 9, the stuff coming off the Dover should be CMR 9 scaling up to be 12 points in height, which is not the same as CMR 12.

The real truth is that fonts produced by Metafont have two different sizes: their *logical size*, and their *physical size*. The logical size is the size that Metafont should have in mind when it decides how to draw the character; the physical size is the actual height of the font as it comes off the printer. For the purposes of this memo, let's measure logical and physical size in *logical points* and *physical points* respectively. Note that printing with a font with sizes 10 logical points and 20 physical points, and then photographically reducing by 50%, is a way to make a Dover give you 768 pixels per logical inch instead of only 384. Of course, the price that you pay is that the Dover now prints on paper that is only 4.25 by 5.5 logical inches!

This conflict between logical size and physical size was already felt with the old Tex fonts, the ones currently on Clover. In addition to the regular CMR, there is a family called SLIDESCMR; the font SLIDESCMR 10 has sizes 10 logical points, and 14 physical points. Note that what has happened here, and in the TimesRoman and TimesRomanD example, is that one kind of size information is being encoded as part of the family name of the font. You have to do something a little bit exotic, since PARC font software only has one size field, and encoding in the family name is one easy thing. In the case of TimesRomanD, the size field gives the physical size, while the presence or absence of a D in the family name tells something about the logical size. In the case of SLIDESCMR, however, this convention is reversed: the size field tells you the logical size while the family name indicates the physical size.

Once the problem is described in these terms, it seems clear that it would be better to make a uniform convention about the use of the single size field provided by PARC software. We hereby propose that the size field should be used exclusively for the physical size of the associated font. Thus, TimesRoman and TimesRomanD are retroactively declared to be doing the right thing, while SLIDESCMR is retroactively declared a mistake. This convention has several advantages. For one thing, it makes it possible to produce a width table for CMR in 10 logical points that can be scaled in the standard PARC way to give the widths for any physical size. In addition, when the printing servers do font substitution, their matching algorithms assume that the size field really describes how big the font will be when coming off the printer.

We will take it as given, then, that the PARC size field for a font shall be used for the physical size. Note that this physical size need not be an integral number of points by the way. Suppose that you are setting math with Tex, using the standard logical sizes of 10, 7, and 5 points. If you want to magnify your output so that it can later be reduced by 25%, you want to use physical sizes 13.33, 9.33, and 6.67 points respectively. These fractional point sizes are likely to confuse humans, but, fortunately, they don't confuse most PARC programs: font sizes are actually measured by the software in micas instead of points, and micas are small enough that rounding to the nearest mica is irrelevant. Unfortunately, PressEdit is a source of difficulty here, since PressEdit does not implement the feature of Press file format which allows font sizes to be specified in micas. We will either fix PressEdit or implement a functional substitute before the TEX cataclysm occurs.

This example of fractional sizes above indicates that the user of Tex doesn't want to be concerned with the physical sizes of fonts. We propose that instead, the Tex user will specify the logical sizes and a magnification factor, where the magnification factor is a floating point number that expresses the ratio between physical and logical size. We plan to generate varying sets of TEX fonts in

varying magnifications: a full set at 1.0 (which is honest, afterall, and the universities would like); a fulll set at 1.1 (which makes CMR 10 the same physical size as TimesRoman 10), a partial set at 1.32 (which makes CMR 10 like TimesRoman 12), and a partial set at 1.54 (which makes CMR 10 like TimesRoman 14). The Tex user will specify the magnification factor by using the magnification parameter of TEX which is built-in as parameter 13, and accessible through "\chpar13". Thus, when a user specifies a font with the command "\font a←CMR10", the system will choose the family CMR in 10 logical points and (10*magnification) physical points. There will also by a "\truefont" macro that assumes a magnification of 1.0, just as the true distance units do.

Having chosen to put the physical size of the Tex fonts in the PARC size field, we are left with the job of encoding the family, face, and logical size of the Tex fonts in the remaining two positions: the PARC family name and the PARC face byte. What are these remaining PARC fields like? Well, the family name is a string that is mapped by an index at the beginning of each font dictionary or Press file into a corresponding eight bit code. The face byte is an eight bit field, of which PARC fonts currently use only the 18 different values from 0 through 21b, where these 18 possibilities correspond via a fixed mapping with the 18 different possible combinations of:
[Light, Medium, or Bold][Regular or Italic][Condensed, Regular, or Expanded].
(Update: A recent change suggested by CMU has added one more component to the face byte field: a three-valued component that distinguishes between different possible character code conventions. Thus, the 54 different face bytes in the range [0,53]=[0b,65b] are now spoken for.]

We can encode information in the PARC family name by just putting it into the string. Unfortunately, if we encode the Tex family, face, and logical size all in the family name, we will generate a huge number of family names; this could become a severe problem in the future, since no font dictionary can talk about more than 256 different families. This desire to limit the explosion of family names suggests that we should try and use the PARC face byte for something.

The first possibility that comes to mind is to use the PARC face byte to encode the Tex face. Unfortunately, Tex fonts come in all sorts of (by PARC standards) bizarre faces, including "Slanted Sans Serif Quotation", "Unslanted Italic", and the like. Thus, to encode the Tex face in the PARC face byte, it would be necessary to extend the fixed PARC face code. Given the arbitrariness of the Tex faces, the only way to extend the code would be to start a list somewhere, allocating face byte values to Tex faces as they arise. That list would become a critical sheet of paper, by the way, since any face code once assigned can never be changed, and can never be used for any other purpose. In fact, it seems undesirable to have to associate code values with arbitrary faces statically when the PARC family name mechanism allows codes values to be associated with strings dynamically. But putting the Tex face into the PARC face byte would be a feasible scheme, at least. If we did that, we would be left with the task of encoding the family and the logical size into the family name. Since there are likely to be fewer magnification factors than logical sizes, it would probably be better, actually, to encode the Tex family and magnification factor into the PARC family name: this would suggest family names such as "ComputerModern-1.54" and the like.

At present, however, I am leaning in favor of a different encoding scheme. The old Tex fonts had already chosen to encode the Tex face into the PARC family name, and that seems to work out rather well. Having handled the Tex family and face with the PARC family name, we are left with the chore of shoehorning the Tex logical size into the PARC face byte. I propose that face values in the range [54, 254]=[66b, 376b] be devoted to measuring the logical sizes of Tex fonts: the amount by which the face byte exceeds 54 will be the logical size of the Tex font measured in half-points. (This will allow logical sizes from zero through 100 points inclusive; the face byte value 255 will be left as an escape value.) For an example, the new version of the font Computer Modern Bold at 10 logical points and 14 physical points would be described by PARC software as having family "CMB", size 14 points, and face 74=54+2*10.

This perhaps exotic scheme has several advantages over the first scheme discussed: the Tex face,

which is basically a string, will be handled by the PARC mechanism best equipped to handle strings. The Tex logical size, which is a number, will be stored as a number instead of encoded into a string. Of course, allowing large values for face bytes will have some impact on PARC software, but a quick survey of knowedgeable people indicates that that impact will be smaller than you might think. The program PrePress already knows about these numeric face codes: face byte values in the appropriate range are input and output as floating point sizes in units of points. Both of the printing servers Spruce and Press, it turns out, don't do anything with face codes except compare them for equality; hence, no changes to either of those systems should be necessary. If you are a font wizard, and you see a reason why this encoding scheme won't work, please let me know. If I hear no bitches, it shall be done as I have said.

Compatibility with the Past:

The other major issue to be considered is compatibility with the past. The new Tex fonts are substantially different from the old ones. More than the fine detail of the character shapes has changed. The widths have changed a lot. The feel of the fonts has also changed, subtly in some cases, and blatantly in others: the upper case script alphabet in CMSY, for example, has been drastically redone.

Whenever a font revision occurs that involves changes in character widths, there is the possibility of old Press files not printing properly. Since the old Tex fonts have been around for more than a year now, there are probably a fair number of Press files in existence that depend upon those old fonts. As the standard PARC fonts got thinner and thinner over the years, one mechanism evolved for handling this problem. Those printers that run Spruce can save multiple tables of width information for a font in their font dictionaries. In addition, every Press file is now stamped with its date of creation. When Spruce receives a Press file, it checks this date; it then prints the file using the current rasters for the font, but spacing those rasters according to the widths that were in effect when that Press file was created. As a result, old Press files come out with different character shapes, but the lines still look justified.

Unfortunately, we can't use this multiple widths mechanism to handle the Tex font portion of the upcoming cataclysm because of the consequences of the logical and physical size decisions made above. Even if we could, the fonts have changed enough so that the results of printing an old Press file with the old widths and the new rasters might not satisfy you. But we can't do it anyway: existing Press files that are asking for CMR at 10 logical points and 10 physical points are asking for CMR with size 10 and face 0. According to the new conventions, you must ask instead for CMR with size 10 and face 74=54+2*10. Thus, the old Tex fonts and the new Tex fonts don't match, according to the tests of the printing servers. Therefore, we propose the following: we will simply leave the old fonts on the printer untouched for a while, to achieve some backward compatibility. After this period, the old fonts will go away, so that we can recapture the space that they require. I would guess that the old Tex fonts will stay on Clover for at least a few months after the cataclysm at least, but not for too much longer. I hope that most Tex users have been smart enough to save their Tex sources, so that dealing with old Press files won't be too big a problem. If you are the type of person who throws away Tex sources, consider yourself warned.

Even though the bulk of the old Tex fonts can stay in Clover's data base without any bad consequences other than making it bigger, the SLIDESCM* fonts are a different case. They are using up family names for what is, in the new scheme of things, no good purpose. I would like to throw those fonts away as part of the cataclysm, even though this would constitute a gross incompatibility with the past. My sense is that they have not been extensively used. And of course, the post-cataclysm Tex and Clover will have roughly equivalent fonts that fit into the scheme detailed above. If you object to the sudden disappearance of the SLIDESCM* fonts, please let me know.

One more minor issue to go! In general, it is impossible to use Tex fonts in Bravo. The standard Fonts.Widths file doesn't include the widths for Tex fonts, because they are too bulky to make that desirable. In addition, Tex fonts in general don't come equipped with Alto versions. Since the mathematical symbols in CMSY seemed too important to pass up, Leo Guibas took the 10 point version of the old CMSY and produced an Alto font for it, calling the result SYMBOL. Some undetermined number of Bravo users have now become dependent on the SYMBOL font. The script alphabet that is part of CMSY is one of the things that has changed the most in the new Tex fonts. My current plan is to leave the SYMBOL font alone; after all, someone might actually like those funny script characters!

Printer Dover

Spruce version 11.0 -- spooler version 11.0

File: <Fonts>UNIVERSITYFONTS.PRESS!2

Creation date: July 20, 1980 12:10 AM

For: gorin

3 total sheets = 2 pages, 1 copy.

Problems encountered:
Font LOGO18 substituted for font LOGO24.

Inter-Office Memorandum

To	Whom it May Concern	Date	July 19, 1980
From	Lyle Ramshaw	Location	Palo Alto
Subject	New fonts for the Universities	Organization	CSL

XEROX

Filed on: [Maxc1]<Fonts>UniversityFonts.bravo
[Maxc1]<Fonts>UniversityFonts.press
[Ivy]<Fonts>Memos>UniversityFonts.bravo
[Ivy]<Fonts>Memos>UniversityFonts.press

The July font cataclysm won't have as much impact at the universities as it will at PARC, since the universities have been running with some version of the third generation fonts from the very beginning. But I am planning to take advantage of the cataclysm as a chance to give the universities a new and more complete release of fonts.

Printing fonts:

Besides adding new special characters in the third generation of fonts, Pellar has also broadened the scope of font sizes and faces that he is supporting. For example, the third generation release includes TimesRoman and Helvetica in 6, 7, 8, 9, 10, 11, 12, and 14 points in plain, bold, italic, and bold-italic, and 18 point in plain, bold, and italic. The 14 point fonts in particular should be popular for overhead transparencies and the like.

In addition to some new sizes of old fonts, I also plan to include several random fonts that were not included in the original release, but which the universities might like to have. If anyone has particular requests, please let me know. So far, in answer to requests from Stanford, I have the following comments. I will be glad to provide Sigma in 20; Logo in 24; Math and Hippo in 12, 14, and 18; Sail in 6 and 8; APL in 8 and 10; and Elite in 10. Cream is available in 10 and 12, in all four faces; I might as well include them all.

By the way, this font release will also include the vector drawing fonts that Carnegie-Mellon produced for use with the ReDraw program; this should make it easier for the other universities to get curves without jaggies out of their Dovers.

TEX fonts:

Unfortunately, the July cataclysm did not include any of the updates to the TEX fonts that I was originally planning on. It just seemed smarter to break the cataclysm into two parts, so as to get some part at least to happen as soon as possible.

When I have recovered from the July cataclysm, I will start working on generating new TEX fonts for CSL from Knuth's latest Metafont sources. My current plan is to generate complete sets at magnifications of 1.0 and 1.1, and partial sets at several larger magnifications. When all of this is done, I will add make these new TEX fonts available to the universities as well, in still another font release. The universities will be able to save some work if they just pull and use the TEX fonts that I produce for PARC, rather than producing their own with Metafont. On the other hand, if you pull my fonts, you will be more or less forced to handle the logical and physical sizes of TEX fonts

in the same way that I have chosen to do so. The right way to handle magnification in TEX and have the resulting Press files print on Dovers is a rather subtle thing to see. If you don't like my plans as expressed in the TexFonts memo, or if you don't understand them, then please bug me about them. I hope to start implementing my plan before too long, and plans are always easiest to change while they are still plans.

Alto fonts:

The university liasons do not have access to [Ivy], and hence cannot go to that source for Alto fonts. I have put the Alto fonts that you probably want onto the Maxc directories <AltoFonts> and <Printing>. Most of them are on <AltoFonts>, in fact. I only used <Printing> to solve the following problem: many people seem to be addicted to the Original-style Alto fonts for Helvetica and TimesRoman, rather than the new fonts that Ron Pellar has produced. Furthermore, they are used to finding the Original guys on <AltoFonts>. Thus, for the entire TimesRoman and Helvetica families, and in three other places where there were conflicts (Hippo12, Logo24, and Math10), I put the Pellar version on <Printing> and the Original version (if any) on <AltoFonts>. Since you are young at heart, flexible, and eager to press on to the future, I suggest that you go with the Pellar version wherever there is any choice. If, after reading the AltoFontGuide memo, you decide that you would like access to some other set of Alto fonts, super-thin ones for example, just let me know.

In any case, all of the Alto fonts are stored on Maxc1 in all three possible flavors: .AL, .KS, and .Strike. The .KS format, in case you are unaware, is a new strike format that allows characters to kern; see the KernedStrikes memo on [Maxc1]<Fonts>.

Space on Maxc1:

As font dictionaries get larger and larger, it will become more of a problem to have the university grant font dictionary sitting on Maxc1, eating up disk space. If the university liasons are willing, I would like to set up a scheme at some point whereby I could put these large files on Maxc1 and send out a message about it with the expectation that the universities would pull the file relatively promptly. When all university liasons had given me the word that they had the file safely in their grasp, I would delete the copy of Maxc1. In this way, we could use Maxc1 only as a temporary storage medium, a part of the transportation process.

Random comment on #140:

Rumor has it that the universities have been having some difficulty with the discrepancies between the PARC character code conventions and those in use elsewhere. One small step in the right direction is being taken by the official third generation font release (although this feature may not have made it into the original font set that I sent the universities). In the new TimesRoman and Helvetica fonts, character code #140, which most of the world thinks of as left single quote but which used to be undefined in PARC fonts, now duplicates #7, which is the PARC standard for left single quote. Let's here it for consistency and all that!