# Code Drafting, Part 3: A Larger Character Set

The first article in this series [1] noted that conventional code drafting considers only letters and treats upper- and lowercase letters as equivalent.

There is some basis for considering upper- and lower case letters to be equivalent. Capitalization is largely a matter of convention. In English the first word of a sentence is capitalized, but that does not change the meaning of the word. (Some other languages have different conventions. In German, all nouns are capitalized.)

On the other hand, the capitalization of titles, acronyms, and so fort´h is more significant and provides an argument that, in some cases, upper- and lowercase letters should be considered to be different.

The most glaring problem with conventional code drafting is its disregard for blanks (spaces). Blanks separate words; in their absence, meaning can be ambiguous or lost altogether. The classic example is the difference between therapist and the rapist. Since the main motivation for code drafting is capturing meaning in a weave, ignoring blanks is a serious flaw.

The problem probably stems from the common view that blanks are not characters and hence are insignificant. In the computer culture, a blank is just as much of a character as the letter A. In the absence of context, in the computer culture, "all characters are created equal". The fact that the blank has no associated glyph (graphic) is irrelevant. In fact, many computer characters have no associated glyphs.

The absence of a glyph for the blank is essential for readability of text. But it can be a problem in some contexts, such as tabular lists, where it cannot be "seen". The symbol ␢ sometimes is used for blanks in such contexts.

Another important aspect of blanks in ordinary text, including strings typically chosen for code drafting, is that blanks occur, on average, more frequently than any single letter.

Digits also are absent from standard code-drafting tables. Because of this, strings with digits are not chosen for code drafting, although they otherwise might well be. Consider

The 1876 Centennial Exposition

And it and it's are different, as are many other such words.

All this leads to the consideration of code tables that include all the characters commonly found in text. A natural basis for this is the characters that are available on modern personal computers.

Computer character sets have developed over time. Modern personal computer have 256 different characters as part of their architecture. Internally, characters have numerical values, which provides a way of ordering them. The 256 different characters are traditionally divided into two 128-character sub-sets. The first 128 are called ASCII and the last 128, Extended ASCII. The ASCII character set provides the basis for "plain" text. Over time, glyphs have been assigned to some characters in the Extended ASCII character set. These include letters with diacritical marks, like ç; mathematical symbols, like ∞; ligatures, like æ; and various other characters. (Different fonts provide different glyphs for the same character, but that is beyond the scope of ordinary code drafting.)

The glyphs for the ASCII character set are standard across different personal computer systems, but the glyphs for the Extended ASCII character set are not. See Reference 2.

The first 32 ASCII characters and the last ASCII character are reserved for keyboard control operations and have no glyphs. This leaves 95 characters with glyphs. Ten are digits, 52 are letters (upper- and lowercase) and the remaining 29 are punctuation marks; common arithmetic operators, like +; commercial symbols, like $, and a few less commonly used characters, like \.

It may seem that some of the 95 ASCII characters would never appear in strings chosen for code drafting. But consider

Screw the &*%^$#/@\ dummies!

One can imagine a smiling corporate executive sitting at a desk behind which is an elegant wall hanging that commemorates this sentence. See the (not-so-elegant) weave at the end of this article.

The 95-character subset of ASCII provides an adequate basis for code drafting. There always will be subjects to be commemorated that cannot be represented with these 95 characters or, in fact, any linear string. Consider

$E = mc^2$

and the famous continued fraction [3]

$$\sin(x) = \cfrac{x}{1 + \cfrac{x^2}{(2 \cdot 3 - x^2) + \cfrac{2 \cdot 3x^2}{(4 \cdot 5 - x^2) + \cfrac{4 \cdot 5x^2}{(6 \cdot 7 - x^2) + \dots}}}}$$

Einstein's famous equation can be spelled out. Complex typography is another matter; a subjects for another approach, perhaps.

Back to reality. Code tables for 95 characters for use with 4-shaft overshot are somewhat cumbersome at an average of about 24 characters per shaft, but they nonetheless manageable.

Such code tables can be constructed in many ways. For example, if upper- and lowercase letters are to be considered equivalent, they can be paired, as in

EeDdHh …

As described in the second article in this series [4], balanced code tables are important in obtaining balanced shaft utilization.

While there are extensive studies of *letter* frequencies in large bodies of text, there appear to be none for *character* frequencies.

An analysis of a small body of text nonetheless can be useful. Here is an example, using sayings in the style of Dave Farber [5]. These sayings are known as farberisms [6]. An example is

It sounds like roses to my ears.

Such sayings have the same structural properties as the strings commonly used for code drafting.

Here are character percentages, in decreasing order, for 1,830 farberisms totalling 69,675 characters:

| | |
|---|---|
| ␢ | 17.3304 (blank) |
| e | 9.5127 |
| t | 7.7115 |
| o | 6.8202 |
| a | 5.5844 |
| n | 4.9501 |
| s | 4.8539 |
| i | 4.5611 |
| h | 4.4635 |
| r | 4.1650 |
| l | 3.0455 |
| . | 2.4987 |
| u | 2.3365 |
| d | 2.3336 |
| g | 1.9677 |
| m | 1.8055 |
| f | 1.7595 |
| c | 1.6476 |
| y | 1.6017 |
| w | 1.5601 |
| ' | 1.5601 |
| p | 1.3189 |
| b | 1.2802 |
| k | 1.0965 |
| I | 0.9429 |
| v | 0.6358 |
| T | 0.4635 |
| H | 0.3903 |
| , | 0.1880 |
| D | 0.1679 |
| W | 0.1607 |
| - | 0.1248 |
| j | 0.1248 |
| A | 0.0889 |
| L | 0.0875 |
| Y | 0.0861 |
| ! | 0.0861 |
| S | 0.0832 |
| x | 0.0631 |

| | |
|---|---|
| M | 0.0545 |
| N | 0.0545 |
| P | 0.0531 |
| z | 0.0531 |
| G | 0.0401 |
| q | 0.0387 |
| B | 0.0272 |
| R | 0.0258 |
| ? | 0.0258 |
| C | 0.0258 |
| E | 0.0200 |
| F | 0.0200 |
| O | 0.0186 |
| J | 0.0186 |
| K | 0.0100 |
| ; | 0.0086 |
| 0 | 0.0086 |
| 5 | 0.0057 |
| 2 | 0.0043 |
| 1 | 0.0043 |
| U | 0.0028 |
| % | 0.0028 |
| “ | 0.0028 |
| 9 | 0.0028 |
| V | 0.0028 |
| Z | 0.0028 |
| : | 0.0014 |

Note the high percentage of blanks. To get frequencies, divide by 100.

Not surprisingly, not all of the 95 characters appear in the farberisms — only 66 do. Most of the missing 29 characters can be assumed to occur with very low frequency, and taken to be 0 for all that matters, in larger bodies of text.

In designing balanced code tables, it probably is sufficient to assume that all digits occur with equal frequency, although in fact, 1 and 0 probably occur somewhat more frequently than the other digits.

The frequency of blanks and punctuation marks in the farberisms probably is similar to what would be expected in strings chosen for code drafting.

A four-shaft, 95-character balanced code table, based on these assumptions and using a technique described in the article on balanced code tables [4], is shown below. The characters have been sorted in the order of their internal codes to make them somewhat easier to locate.

Because of the uneven distribution of characters among the shaft sets, this table would be difficult to use without computer assistance. A better method of construction, which balances both the number of chracters per shaft set and the frequencies is needed. A small research project, perhaps.

| *characters* | *shaft pair* | *frequency* |
|---|---|---|
| ␢#$&()*+,/134678<=>@EQX[\\]^_`ov{\|}~ | 1,2 | .249987 |
| DMaegt | 2,3 | .249987 |
| GWhimnrs | 3,4 | .249999 |
| !\”%’-.0259:;?ABCFHIJKLNOPRSTUVYZbcdfjklpquwxyz | 4,1 | .249995 |

## References

1. *Code Drafting, Part 1: Introduction*, Ralph E. Griswold, 2004:
http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_cd1.pdf

2. *Character Sets*, Ralph E and Madge T. Griswold, 2004:
http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_char.pdf

3. *Continued Fraction Sequences and Weave Design, Part 1: Introduction*, Ralph E. Griswold, 2000:
http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_cf1.pdf

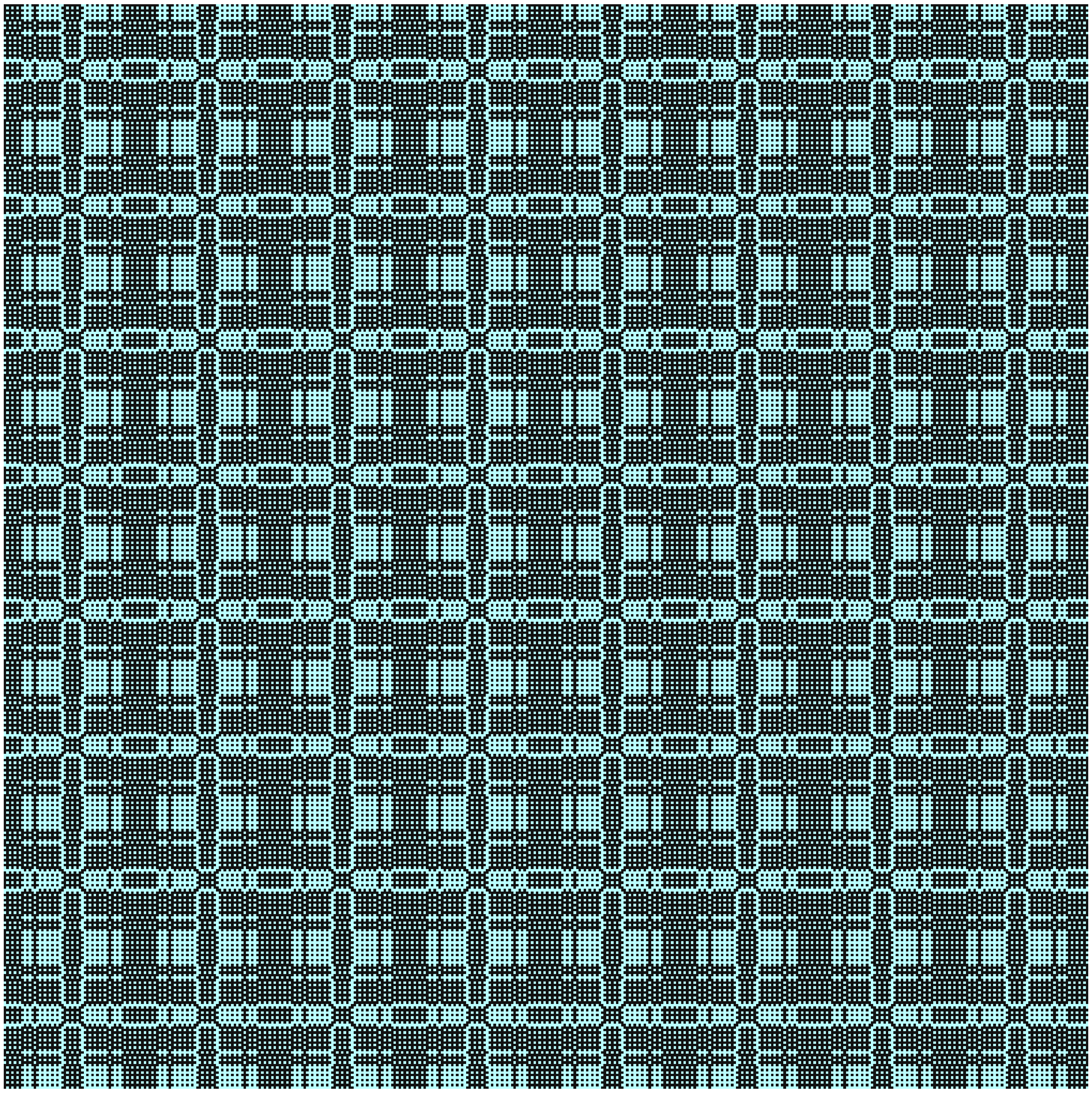
4. *Code Drafting, Part 2: Balanced Code Tables*, Ralph E. Griswold:
http://www.cs.arizona.edu/patterns/weaving/webdocs/gre_cd2.pdf

5. Dave Farber's home page:
http://www.cis.upenn.edu/~farber/

6. Farberisms:
http://www.cs.arizona.edu/icon/oddsends/farber.htm

Ralph E. Griswold

Department of Computer Science
The University of Arizona
Tucson, Arizona



**The CEO's Wall Hanging**