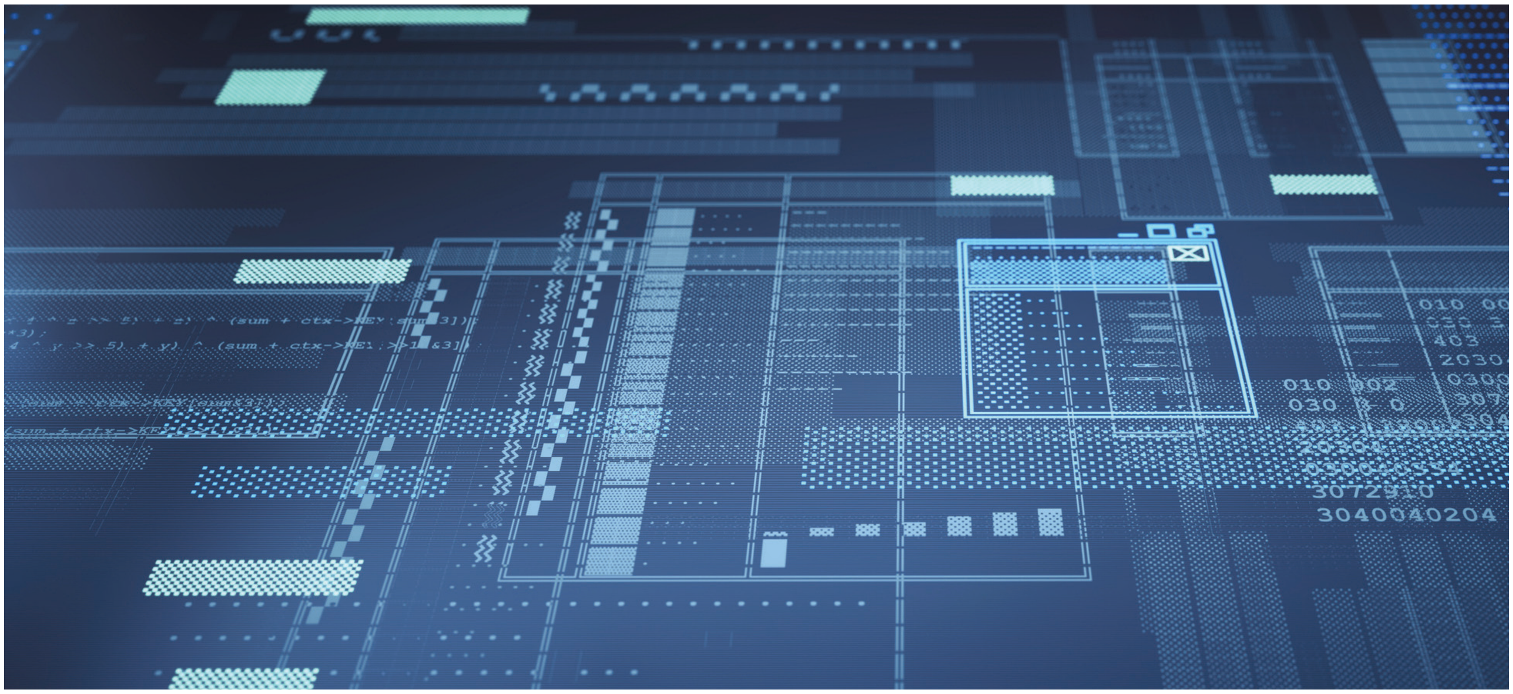

From Gibberish to Unicode

BY DAVID KALAT

With the aggressive pace of technological change and the onslaught of news regarding data breaches, cyber-attacks, and technological threats to privacy and security, it is easy to assume these are fundamentally new threats. The pace of technological change is slower than it feels, and many seemingly new categories of threats have been with us longer than we remember.

Nervous System *is a monthly series that approaches issues of data privacy and cyber security from the context of history—to look to the past for clues about how to interpret the present and prepare for the future.*

In an early landmark of eDiscovery case law, *CP Solutions PTE, Ltd. v. General Elec. Co.* (D. Conn. Feb. 6, 2006), the plaintiff objected to a variety of alleged defects in defendants' production, including the production of supposedly thousands of pages of "gibberish." The court ruled that, to the extent that the underlying documents were created or received by any of the defendants in a readable format, they must be produced for plaintiff in a readable, usable format.

From a standard of jurisprudence this seems an imminently reasonable conclusion. More interesting, though, is the technical question of why the gibberish appeared at all. While many system files and non-text-based electronic documents routinely appear as "gibberish" when rendered as printed text, an entire category of genuine written text-based communication can, in certain conditions, end up appearing as a nonsensical jumble of odd symbols and unreadable characters. To understand why this happens, and to solve for it, requires looking under the hood of how binary data handles text in the first place.

When computers were first gaining widespread use in American businesses in the 1960s and 1970s, the standard byte length was 7 bits. This allowed for 128 possible values per 7-bit byte. For the kinds of applications then in use, this was sufficient to cover the entire 26-letter English alphabet in both upper and lowercase, the numerical values 0 through 9, a variety of punctuation and symbols, and some legacy control codes left over from the era of Teletype machines. This system of mapping the 128 integers available in 7-bit computing to each of these unique values is called ASCII (American Standard Code for Information Interchange).

Although ASCII provided the necessary mapping of binary byte values to standard English writing, it made no provision for documents to contain non-English text. No characters were available for any other alphabets.

With the advent of 8-bit computing, 128 values were added (because each additional bit doubles the available values). To maintain backward compatibility, the original ASCII character set was carried over. This meant that the first 128 values still mapped to the same ASCII characters as before, but the additional set of 128 values could be used for additional characters—here is where things got messy.

Different computing systems deployed the additional 128 values in different ways. The DOS operating system used the extra values for smiley faces and other graphic symbols; Macintosh computers used the values for a mix of typographic marks and international letters; the "Multinational Character Set" by Digital Equipment Corporation focused on international alphabets.

Foreign language character support still proved problematic. The 128 values were insufficient to encompass the wide array of characters needed to support every foreign language—Japanese, Chinese, Greek, Arabic, Hebrew, and so on. The Japanese language alone has roughly 2,000 written characters that are officially designated for study by high schoolers, out of an even larger pool of less commonly used characters.

As a result, the characters represented by the second set of 128 values could be read by different computer systems in very different ways. What might appear as a bunch of clip art characters on one computer could turn into foreign language characters on another, or vice versa. This made the interchange of documents among users in different countries—even among users of different operating systems—vexing.

One possible "solution" would be to assign 3 bytes to every character. That would allow for over 16.7 million different characters, which would be more than sufficient to cover all alphabets. This posed its own problem, however. Users who only need to write in English only need 7 or 8 bits, so implementing a 3-byte length per character would result in significant waste of data storage space for many users.

Unicode was developed as an industry-standard solution for this problem. A universal code chart was published that mapped every unique character of every human language, along with all necessary numerical characters, punctuation, typographical marks, and accent marks. Of course, this code map encompassed far more than 256 values. In order to provide maximum compatibility with existing systems, the first 128 Unicode values match the original ASCII values. Past those values, Unicode allows for the use of multiple bytes to represent the additional character sets, but only when necessary. English text is encoded using only 8 bits.

While situations such as the so-called "gibberish" production encountered in *CP Solutions PTE, Ltd. v. General Elec. Co.* are now thankfully rare, the problem can still arise any time a document is encoded using a character set different from the one used to display or print that document. Not all documents are encoded in Unicode, and the use of nonstandard character sets can still be encountered.

The views and opinions expressed in this article are those of the author and do not necessarily reflect the opinions, position, or policy of Berkeley Research Group, LLC or its other employees and affiliates.

Legaltech© news