Some Boring Stuff You Need To Understand Before You Can Dive In

Few people think about it, but text is incredibly complicated. Start with the alphabet. The people of Bougainville have the smallest alphabet in the world; their Rotokas alphabet is composed of only 12 letters: A, E, G, I, K, O, P, R, S, T, U, and V. On the other end of the spectrum, languages like Chinese, Japanese, and Korean have thousands of characters. English, of course, has 26 letters — 52 if you count uppercase and lowercase separately — plus a handful of *!@#$%&* punctuation marks.

When you talk about "text," you're probably thinking of "characters and symbols on my computer screen." But computers don't deal in characters and symbols; they deal in bits and bytes. Every piece of text you've ever seen on a computer screen is actually stored in a particular `character encoding`. Very roughly speaking, the character encoding provides a mapping between the stuff you see on your screen and the stuff your computer actually stores in memory and on disk. There are many different character encodings, some optimized for particular languages like Russian or Chinese or English, and others that can be used for multiple languages.

In reality, it's more complicated than that. Many characters are common to multiple encodings, but each encoding may use a different sequence of bytes to actually store those characters in memory or on disk. So you can think of the character encoding as a kind of decryption key. Whenever someone gives you a sequence of bytes — a file, a web page, whatever — and claims it's "text," you need to know what character encoding they used so you can decode the bytes into characters. If they give you the wrong key or no key at all, you're left with the unenviable task of cracking the code yourself. Chances are you'll get it wrong, and the result will be gibberish.

Everything you thought you knew about strings is wrong.

Surely you’ve seen web pages like this, with strange question-mark-like characters where apostrophes should be. That usually means the page author didn’t declare their character encoding correctly, your browser was left guessing, and the result was a mix of expected and unexpected characters. In English it’s merely annoying; in other languages, the result can be completely unreadable.

There are character encodings for each major language in the world. Since each language is different, and memory and disk space have historically been expensive, each character encoding is optimized for a particular language. By that, I mean each encoding using the same numbers (0–255) to represent that language’s characters. For instance, you’re probably familiar with the ascii encoding, which stores English characters as numbers ranging from 0 to 127. (65 is capital “A”, 97 is lowercase “a”, &c.) English has a very simple alphabet, so it can be completely expressed in less than 128 numbers. For those of you who can count in base 2, that’s 7 out of the 8 bits in a byte.

Western European languages like French, Spanish, and German have more letters than English. Or, more precisely, they have letters combined with various diacritical marks, like the ñ character in Spanish. The most common encoding for these languages is CP-1252, also called “windows-1252” because it is widely used on Microsoft Windows. The CP-1252 encoding shares characters with ascii in the 0–127 range, but then extends into the 128–255 range for characters like n-with-a-tilde-over-it (241), u-with-two-dots-over-it (252), &c. It’s still a single-byte encoding, though; the highest possible number, 255, still fits in one byte.

Then there are languages like Chinese, Japanese, and Korean, which have so many characters that they require multiple-byte character sets. That is, each “character” is represented by a two-byte number from 0–65535. But different multi-byte encodings still share the same problem as different single-byte encodings, namely that they each use the same numbers to mean different things. It’s just that the range of numbers is broader, because there are many more characters to represent.

That was mostly OK in a non-networked world, where “text” was something you typed yourself and occasionally printed. There wasn’t much “plain text”.

Source code was ascii, and everyone else used word processors, which defined their own (non-text) formats that tracked character encoding information along with rich styling, &c. People read these documents with the same word processing program as the original author, so everything worked, more or less.

Now think about the rise of global networks like email and the web. Lots of “plain text” flying around the globe, being authored on one computer, transmitted through a second computer, and received and displayed by a third computer. Computers can only see numbers, but the numbers could mean different things. Oh no! What to do? Well, systems had to be designed to carry encoding information along with every piece of “plain text.” Remember, it’s the decryption key that maps computer-readable numbers to human-readable characters. A missing decryption key means garbled text, gibberish, or worse.

Now think about trying to store multiple pieces of text in the same place, like in the same database table that holds all the email you’ve ever received. You still need to store the character encoding alongside each piece of text so you can display it properly. Think that’s hard? Try searching your email database, which means converting between multiple encodings on the fly. Doesn’t that sound fun?

Now think about the possibility of multilingual documents, where characters from several languages are next to each other in the same document. (Hint: programs that tried to do this typically used escape codes to switch “modes.” Poof, you’re in Russian koi8-r mode, so 241 means Я; poof, now you’re in Mac Greek mode, so 241 means ώ.) And of course you’ll want to search those documents, too.

Now cry a lot, because everything you thought you knew about strings is wrong, and there ain’t no such thing as “plain text.”