UTF-8

UTF-8 uses 8-bit code units, and it represents characters in the Basic Latin (ASCII) range U+0000 to U+007F efficiently, one code unit per character. On the other hand, this implies that all other characters use at least two code units, which all have the most significant bit set—i.e., they are in the range 80 to FF (hexadecimal). More exactly, they are in the range 80 to 9F. This means that when there is a code unit in the range 00 to 7F in UTF-8 data, we can know that it represents a Basic Latin character and cannot be part of the representation of some other character.

These structural decisions imply that UTF-8 is relatively inefficient, since it leaves many simple combinations unused. There is yet another principle that has a similar effect. In a representation of any character other than Basic Latin characters, the first (leading) code unit is from a specific range, and all the subsequent (trailing) code units are from a different range.

UTF-8 Encoding Algorithm

For a character outside the Basic Latin block, UTF-8 uses two, three, or four octets. You might encounter specifications that describe UTF-8 as using up to six octets per character, but they reflect definitions that did not restrict the Unicode coding space the way it has now been restricted.

The UTF-8 algorithm is described in Table 6-1. The first column specifies a bit pattern, in 16 or 21 bits, grouped for readability. The other columns indicate how the pattern is mapped to code units (octets), represented here as bit patterns.

Table 6-1. UTF-8 encoding algorithm

00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
uuuww zzzzyyyy yyxxxxxx	11110uuu	10wwzzzz	10yyyyyy	10xxxxxx

Thus, the UTF-8 encoding uses bit combinations of very specific types in the octets. If you pick up an octet from UTF-8 encoded data, you can immediately see its role. If the first bit is 0, the octet is a single-octet representation of a (Basic Latin) character. Otherwise, you look at the second bit as well. If it is 0, you know that you have a second, third, or fourth octet of a multioctet representation of a character. Otherwise, you have the first octet of such a representation, and the initial bits 110, 1110, or 1111 reveal whether the representation is two, three, or four octets long.

Thus, interpreting (decoding) UTF-8 is straightforward, too. You take an octet, match it with the patterns in column "Octet 1" in Table 6-1, and read zero to three additional octets accordingly. Then you construct the binary representation of the code number from the