

Typing Emojis in a Hex Editor: A Guide to Bytes, UTF, and Endianness

This guide will show you how to build a text file completely from scratch using raw hexadecimal (hex) numbers. We will write a Grinning Face (😊), a space, and a Rocket (🚀), and explore why the numbers change depending on whether you are building a local document or a website.

1. How to Open a Hex Editor

A standard text editor (like Notepad or Word) hides the raw computer data from you. A **hex editor** lets you see and type the exact 1s and 0s (represented as base-16 hex numbers) that make up files on your computer.

Getting Started:

The easiest way to do this without installing any software is to use a web-based hex editor.

1. Open your web browser and go to hexed.it.
2. Click **New file** at the top.
3. You will see a grid of 00s or a completely empty workspace. This is your raw canvas.

2. What to Type: Document vs. Website

When you type in a hex editor, you are typing raw bytes. The exact bytes you type depend on the **encoding standard** you want to use.

Option A: The Website Standard (UTF-8)

If you are building a website (.html file) or a modern text file, the standard is **UTF-8**.

Click into the hex grid and type these exact numbers (spaces are usually added automatically by the editor):

F0 9F 98 80 20 F0 9F 9A 80

Breakdown:

- F0 9F 98 80 = 😊 (Grinning Face)
- 20 = Space
- F0 9F 9A 80 = 🚀 (Rocket)

How to test: Save/Export the file as emojis.html and drag it into your web browser. You will see:



Option B: The Legacy Windows Document Standard (UTF-16 Little-Endian)

If you are building a classic Windows text document (where Windows native APIs do the heavy lifting), it relies on **UTF-16 Little-Endian (UTF-16LE)**.

Type these exact numbers instead:

FF FE 3D D8 00 DE 20 00 3D D8 80 DE

Breakdown:

- FF FE = The "Byte Order Mark" (Tells the computer: "Hey, this is a UTF-16LE document!")
- 3D D8 00 DE = 😄 (Grinning Face)
- 20 00 = Space
- 3D D8 80 DE = 🚀 (Rocket)

How to test: Save/Export the file as emojis.txt and open it in Windows Notepad. You will see: 😄



3. What do the "8" and "16" mean in UTF?

Unicode is a giant dictionary assigning a specific number to every character in the world (e.g., the rocket is ^{128,640}, or ^{0x1F680}). **UTF** (Unicode Transformation Format) is the rulebook for how to store that number in computer memory.

The numbers **8** and **16** refer to the minimum **Code Unit** the computer uses to build a character. You can think of Code Units like building blocks:

- **UTF-8 uses ⁸-bit Code Units (1 byte).**
Standard letters like 'A' fit inside one unit (e.g., 41). Complex emojis require up to four units stuck together (e.g., F0 9F 9A 80). Because the units are small, UTF-8 is highly efficient for English text and is the undisputed king of the internet.

- **UTF-16 uses ¹⁶-bit Code Units (2 bytes).**
Every standard character gets a minimum of one ¹⁶-bit unit (e.g., 'A' is 00 41). For emojis that are too big for one unit, UTF-16 uses a "Surrogate Pair"—two ¹⁶-bit units side-by-side.

4. Endianness: Why do the bytes flip?

If you compare the hex codes from earlier, you might notice something weird happens in UTF-16.

The mathematical hex code for the Rocket Emoji's surrogate pair is:

^{0xD83D} and ^{0xDE80}

But in the Windows document, we typed: 3D D8 and 80 DE.

This happens because of **Endianness**—specifically, **Little-Endian** architecture, which is how Intel and AMD processors read memory.

- **Big-Endian:** Reads numbers left-to-right, just like humans do. The biggest (most significant) part of the number comes first.
- **Little-Endian:** Stores the smallest (least significant) byte first.

Because UTF-16 groups data into ¹⁶-bit (²-byte) code units, a Little-Endian computer takes each unit and stores it backwards in memory to process it faster.

- Unit 1: ^{0xD83D} flips to **3D D8**

- Unit 2: **0xDE80** flips to **80 DE**

Why doesn't this happen to Websites (UTF-8)?

UTF-8 operates purely on ⁸-bit code units (1 byte). Endianness only applies when a computer has to look at a chunk of data larger than ¹ byte at a time.

Because UTF-8 handles everything one single byte at a time, there is no "left" or "right" side of a byte to flip. A byte is just a byte. The computer sends F0, then it sends 9F, then 9A, then 80. This makes UTF-8 "endian-neutral," which is why it is perfect for websites where you don't know if the person loading the page is using a PC, a Mac, or a mobile phone!