Daasdt

Character Encoding Schema

UniCode

Daasdt

by Daasdt (RV)

Table of Contents

Introduction to Character Encoding Standard

UniCode

01

- Universal Char Encoding Standard
- Almost All World Characters
- UTF-8 , UTF-16 and UTF-32 (formats)
- ASCII , ISO8859-1 – Other Schemas

- An AI Default Character schema

Daasdt

by Daasdt (RV)

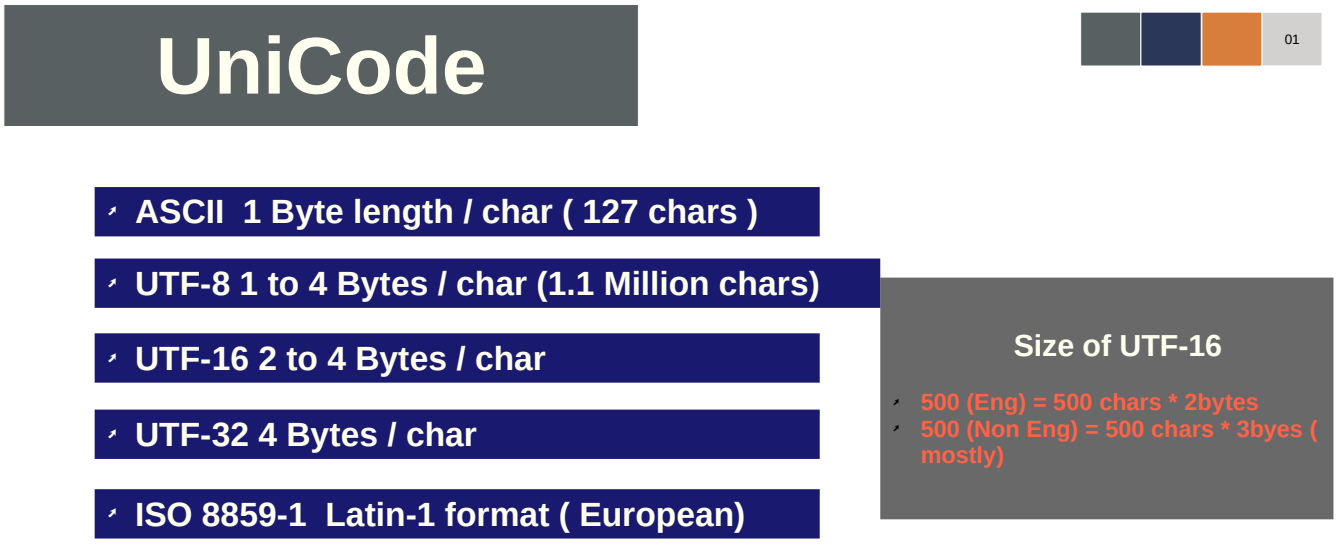
Unicode is a universal character encoding standard used to store and transmit text across IT systems. It supports over 1.1 million characters, allowing for the representation of nearly all written languages and symbols used worldwide.

In contrast, ASCII (American Standard Code for Information Interchange) is the earliest character encoding scheme, designed to store only 128 characters, mainly for English letters and basic punctuation. It was limited to representing English text.

Unicode, however, assigns a unique “code point” to each character (for example, U+0041 represents the letter ‘A’), and these code points remain the same regardless of the encoding format. There are three main encoding formats for Unicode: “UTF-8”, “UTF-16”, and “UTF-32”, each using a different method to represent these code points in memory.

A key point is that most AI systems, by default, use Unicode encoding, ensuring compatibility with multiple languages and character sets.

Storage of Character Encoding Standard



Each character in “ASCII” is stored using “Single byte”, allowing the schema to support only 128 characters, which are directly mapped into the system’s memory in a fixed 1-byte format.

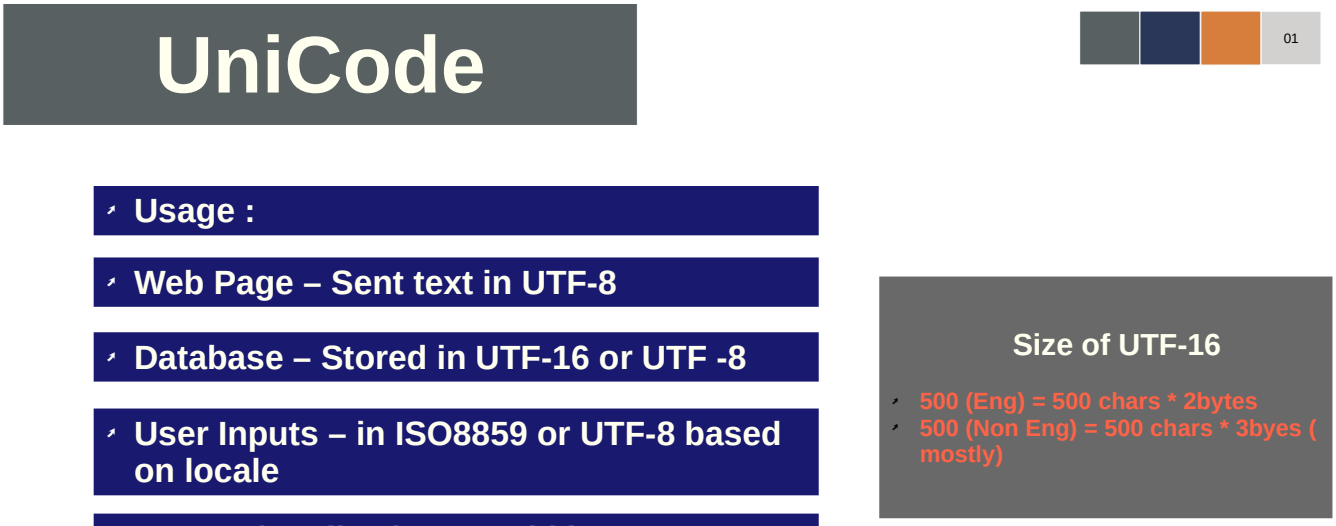
“UTF-8” uses a *“variable-length encoding”* format. The first 128 characters (which correspond to ASCII values) are stored in “1 byte”. Characters beyond the ASCII range are encoded using “2 to 4 bytes”, depending on their position in the “Basic Multilingual Plane (BMP)”.

“UTF-16” stores characters from the BMP in “2 bytes”, while characters outside the BMP require “4 bytes”.

“UTF-32” stores every character in a fixed “4 bytes”, making it less memory-efficient but ensuring uniform byte length for all characters.

The “Latin-1” encoding schema, also known as “ISO-8859-1”, was specifically developed to support “European languages” with characters common in Western European alphabets.

Usage of Character Encoding Standard



For size calculation, you can estimate the storage requirements by considering the following:

- If the text contains “English characters” (which are mostly within the ASCII range), each character typically takes “2 bytes”.
- For other characters (such as those from non-Latin scripts), it’s safe to assume an average of “3 bytes” per character. This works for most non-ASCII characters, though some may take fewer or more bytes depending on the encoding.

To calculate the total storage required, multiply the ***number of rows, records, or transactions** by the **calculated field length*** (based on the average byte size per character).

For web pages, “UTF-8” encoding is typically used to send characters, and you can verify this by checking the HTTP payload header.

For databases, the choice between “UTF-16” or “UTF-8” depends on the system architecture. “UTF-16” is often used in systems where 2-byte characters are common, while “UTF-8” is more flexible and efficient for web applications.

Regarding user input, the encoding format will typically depend on the “locale”:

- For English and Western European languages, “Latin-1 (ISO-8859-1)” might be used.
- For broader language support, “UTF-8” is commonly used to ensure compatibility with various scripts.

Conclusion to use Character Encoding Standard

When designing an application with “internationalization” in mind (supporting multiple languages), it’s generally best to choose “UTF-8” encoding, as it is the most flexible and widely supported standard.

For “localization” (adapting to specific regions or languages), you can adjust the character encoding as needed, but UTF-8 provides a solid foundation for handling global text in your application.